



ISSCC 2019

SESSION 7

Machine Learning

An 11.5 TOPS/W 1024-MAC Butterfly-Structure Dual-Core Sparsity-Aware Neural Processing Unit in 8nm Flagship Mobile SoC

Jinook Song¹, Yunkyo Cho¹, Jun-Seok Park¹, Jun-Woo Jang²,
Sehwan Lee², Joon-Ho Song², Jae-Gon Lee¹, Inyup Kang¹

SAMSUNG Exynos

¹Samsung Electronics, Hwaseong, Korea

²Samsung Advanced Institute of Technology, Suwon, Korea

Outline

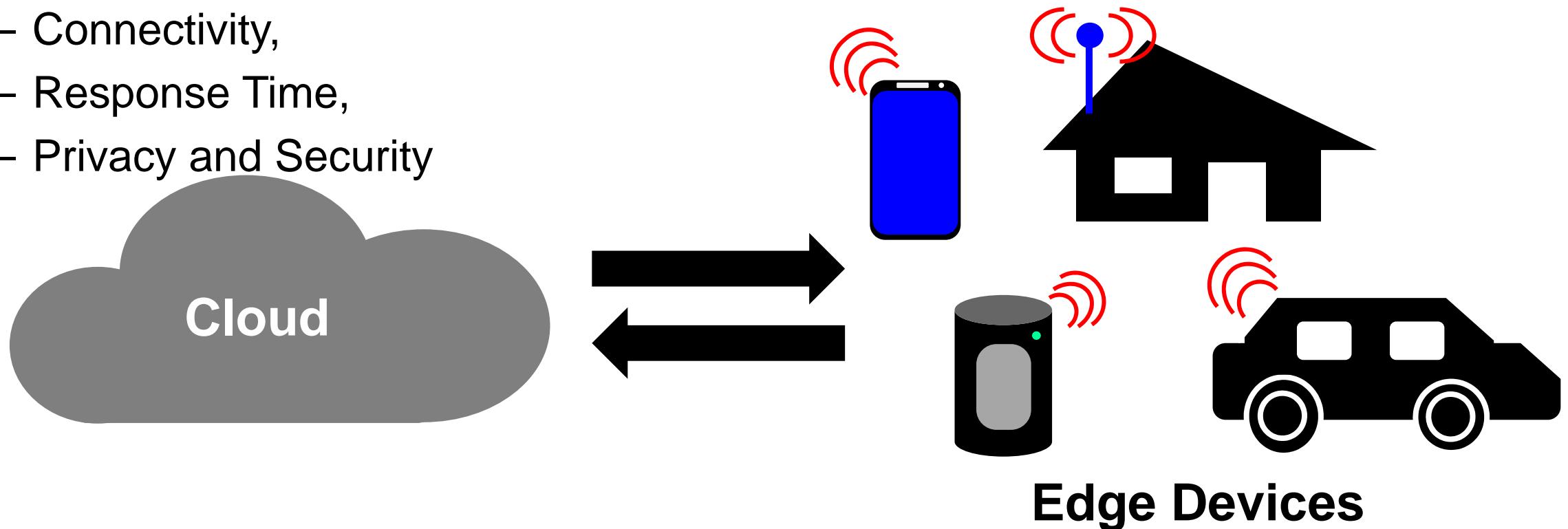
- Motivation
- Background
- Overall Architecture
- Key Features
 - Butterfly-Structure Dual-Core Accelerator
 - Sparsity-Aware Computing
 - Bandwidth-Efficient Network Traversal
- Measurement Results
- Conclusion

Outline

- Motivation
- Background
- Overall Architecture
- Key Features
 - Butterfly-Structure Dual-Core Accelerator
 - Sparsity-Aware Computing
 - Bandwidth-Efficient Network Traversal
- Measurement Results
- Conclusion

Motivation

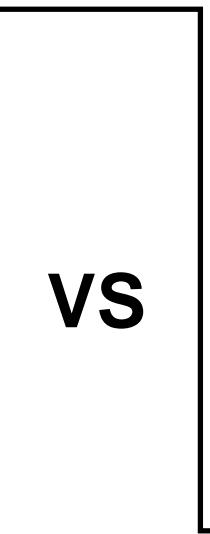
- Deep Learning Application:
 - Voice-Activated Assistants, Automatic Machine Translation, Text Generation, Image Recognition and etc.
- Needs On-Device Solution or Mobile Platform Considering
 - Connectivity,
 - Response Time,
 - Privacy and Security



Motivation

Deep Learning Application

**High Performance
High Memory Bandwidth**



Mobile SoC

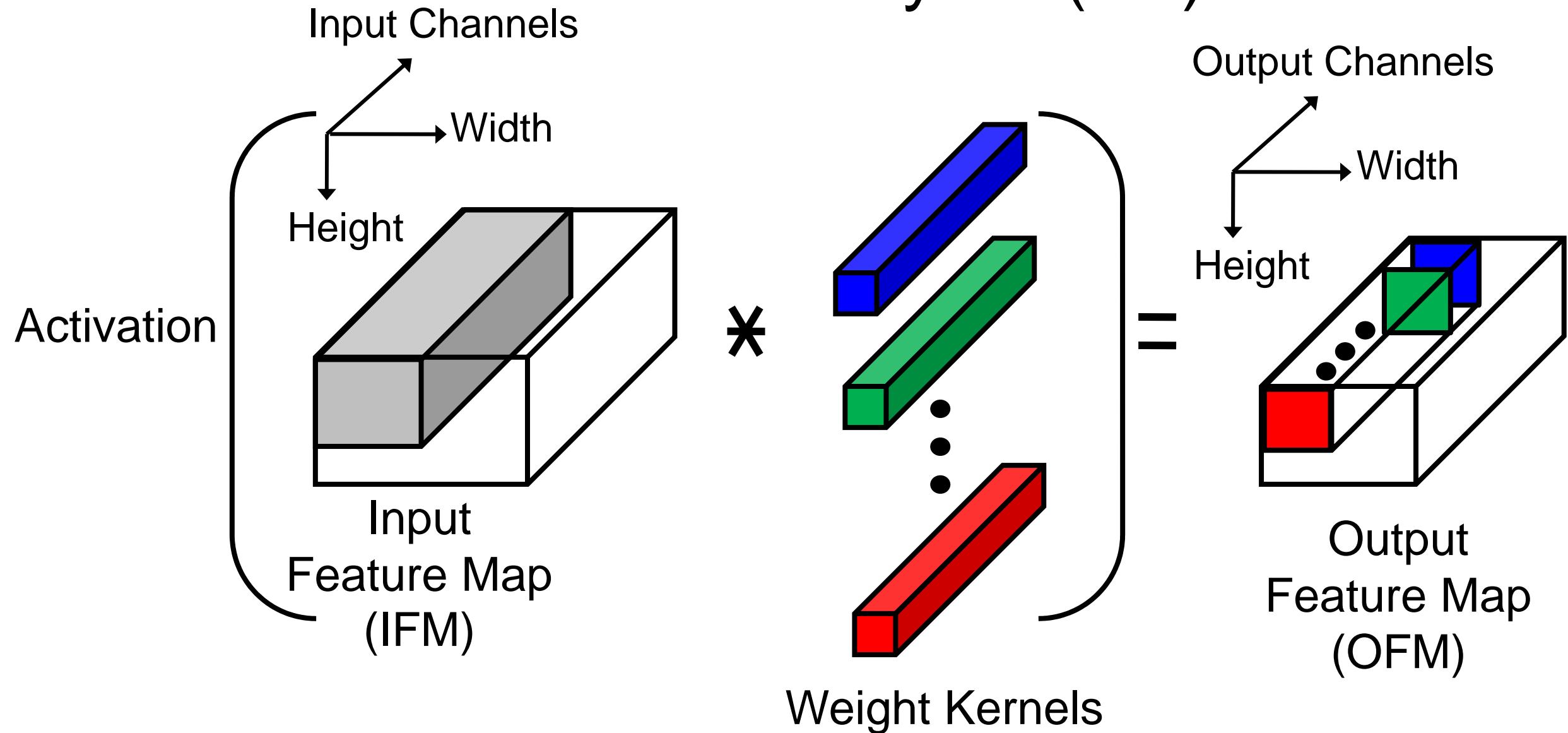
**Small Area
Low Energy
Low Memory Bandwidth**

Mobile SoC Needs an IP with
**High Performance,
Small Area,
Low Energy and
Low Memory Bandwidth**

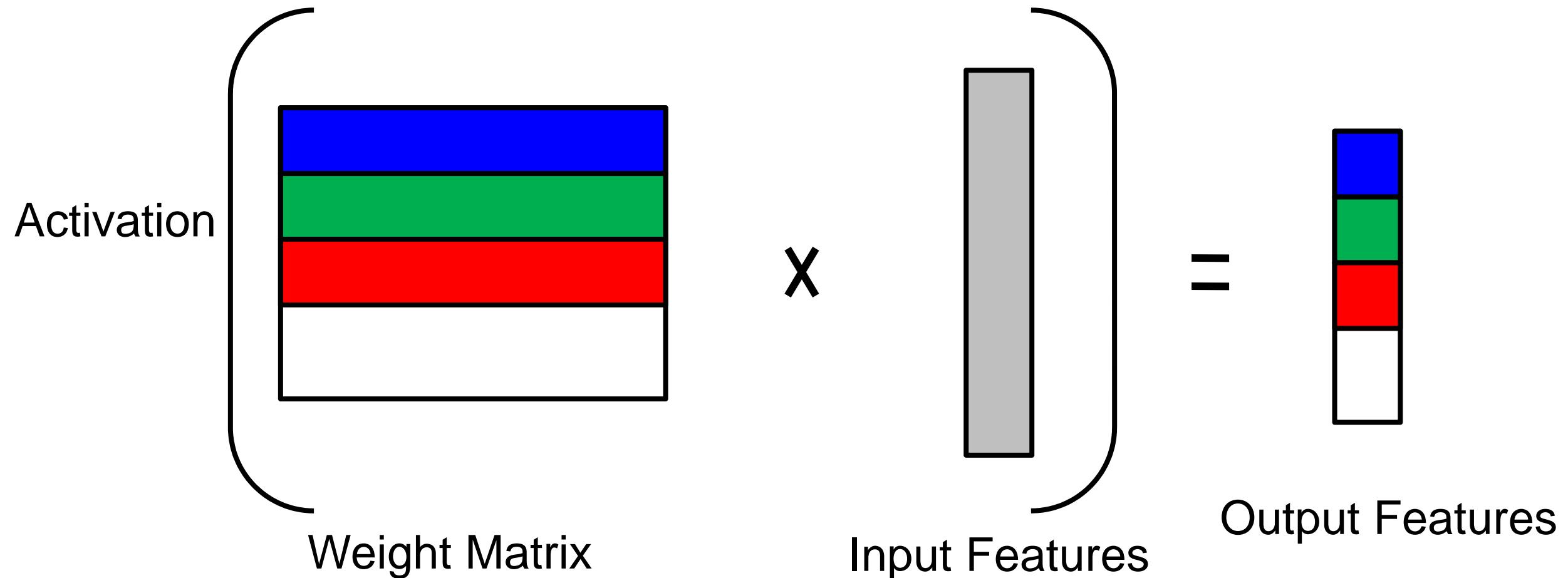
Outline

- Motivation
- Background
- Overall Architecture
- Key Features
 - Butterfly-Structure Dual-Core Accelerator
 - Sparsity-Aware Computing
 - Bandwidth-Efficient Network Traversal
- Measurement Results
- Conclusion

Convolution Layers (CL)

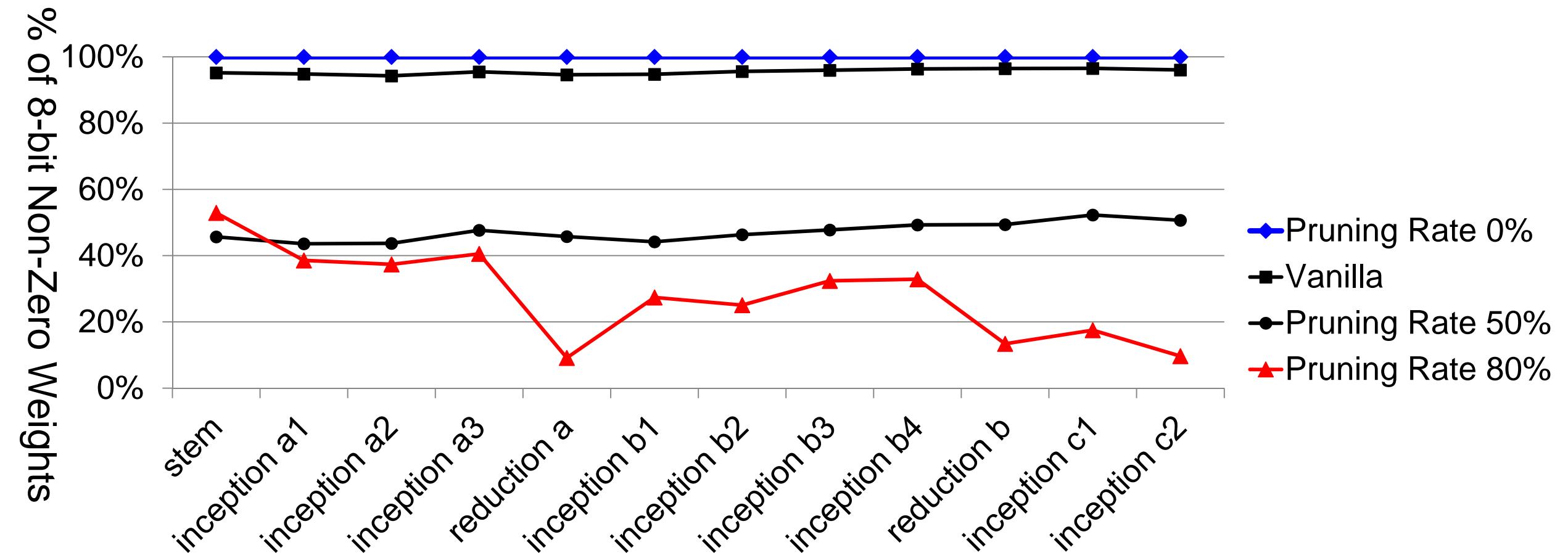


Fully-Connected Layers (FCL)



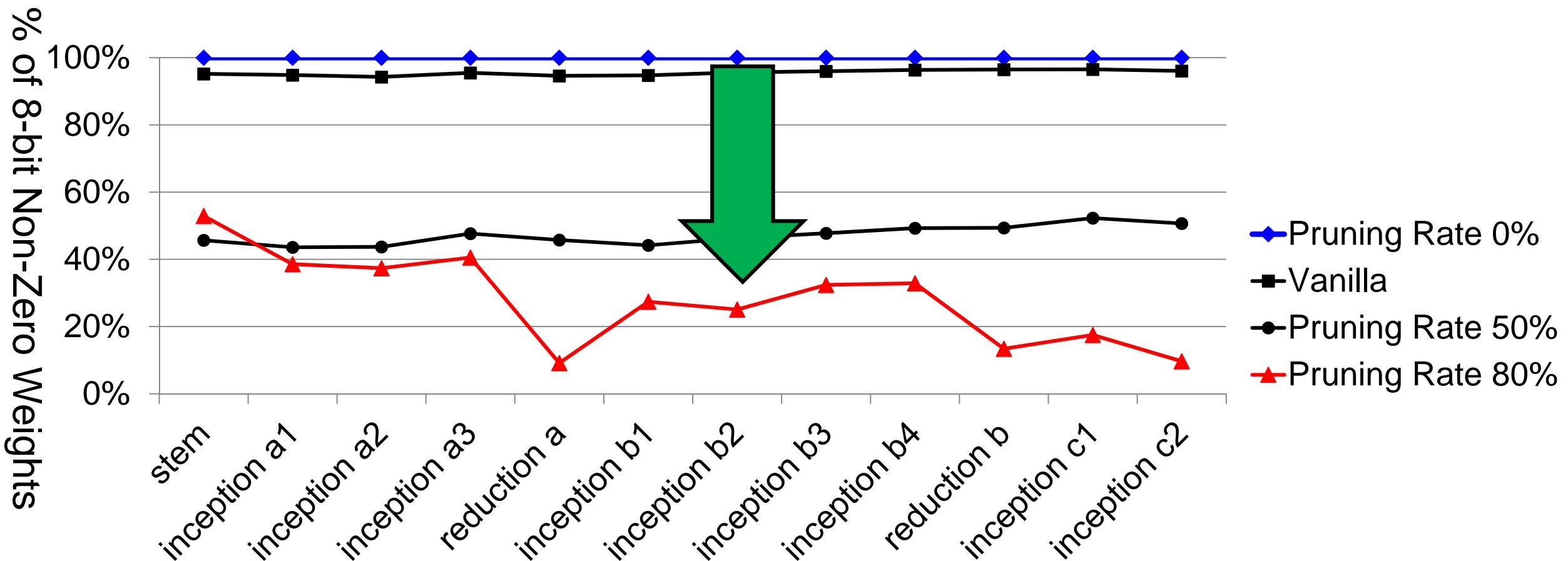
Weight Pruning and 8-bit Quantization

- Less than 1 % accuracy difference



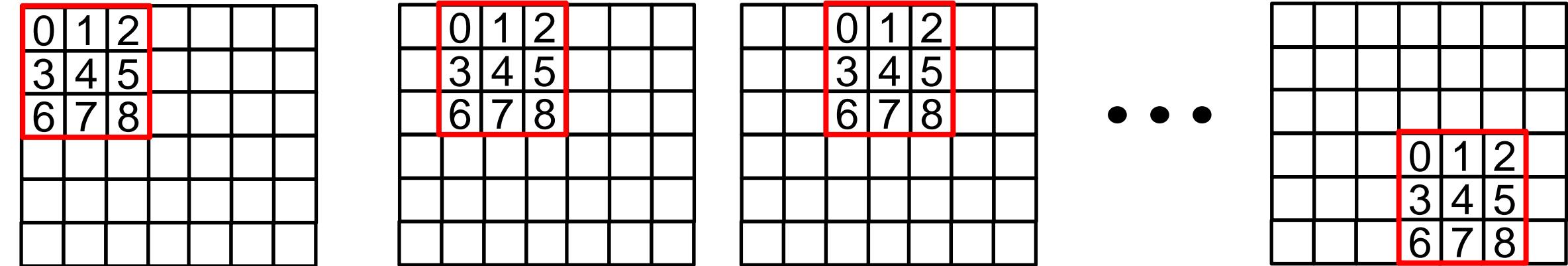
Weight Pruning and 8-bit Quantization

- Reduce the number of multiplying and accumulating (MAC) operations
- Reduce the weight size

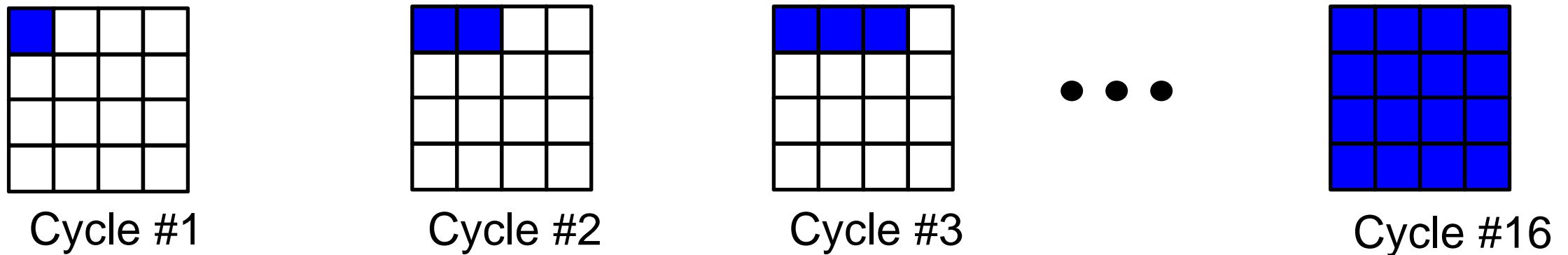


Convolution: Moving an Weight Kernel

Input Feature Map



Output Feature Map



Convolution: Moving Output Feature Maps

Non-zero Weights

0

1

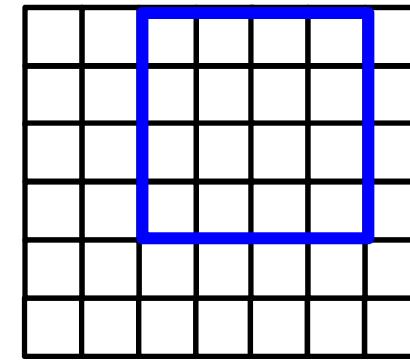
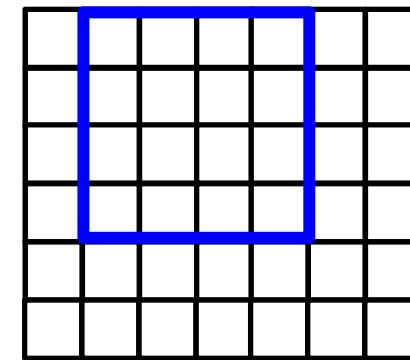
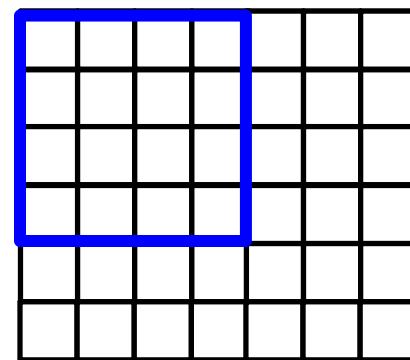
2

8

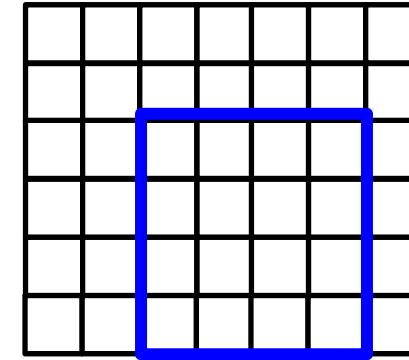
Weight Kernel

0	1	2
3	4	5
6	7	8

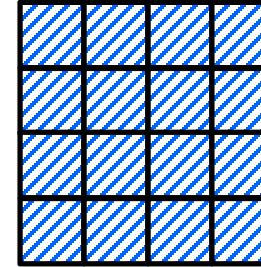
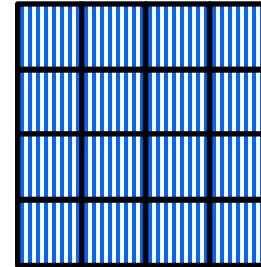
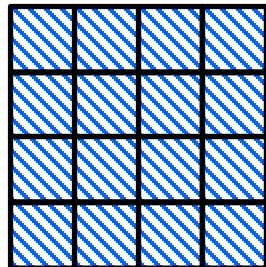
Input Feature Map



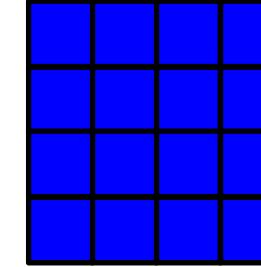
• • •



Output Feature Map



• • •



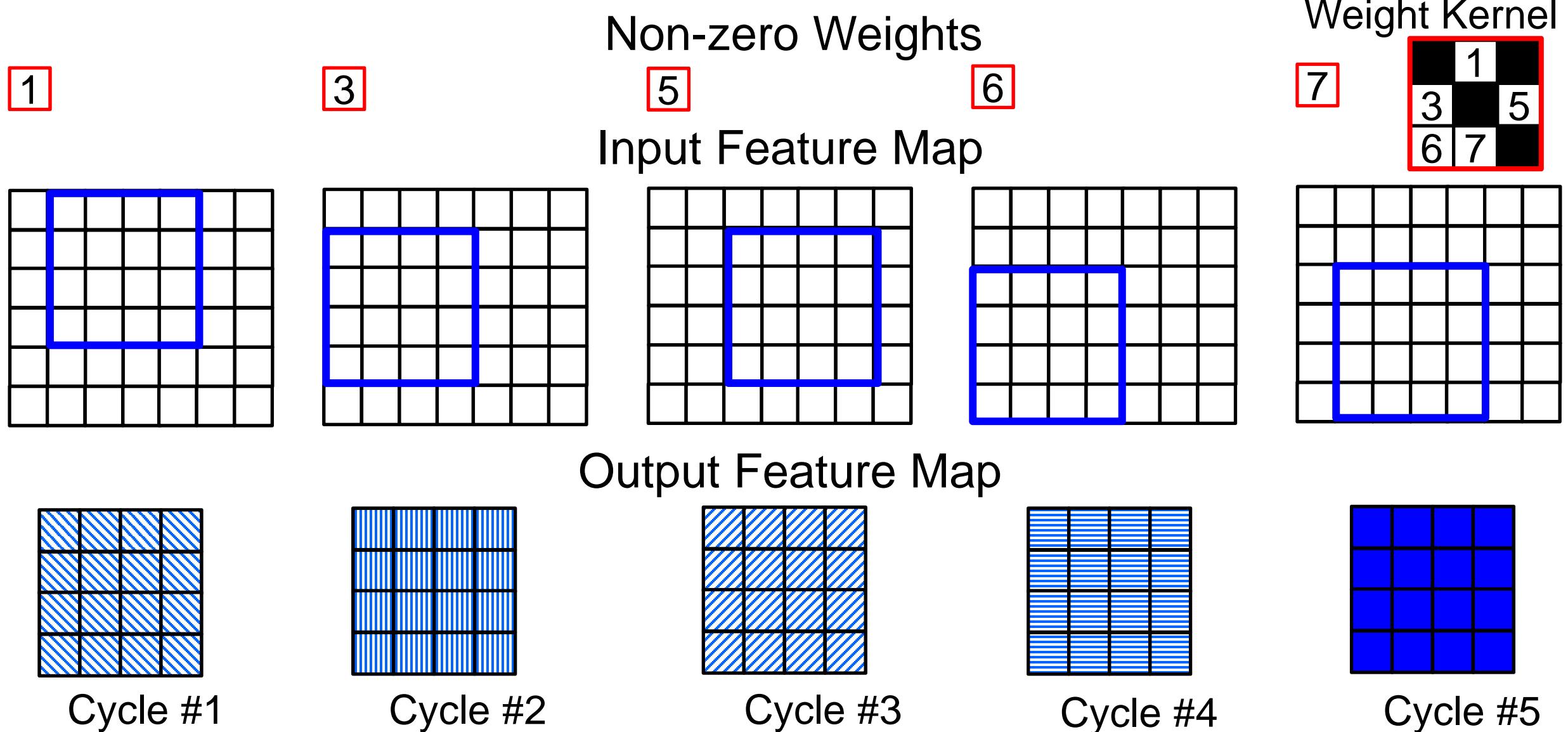
Cycle #1

Cycle #2

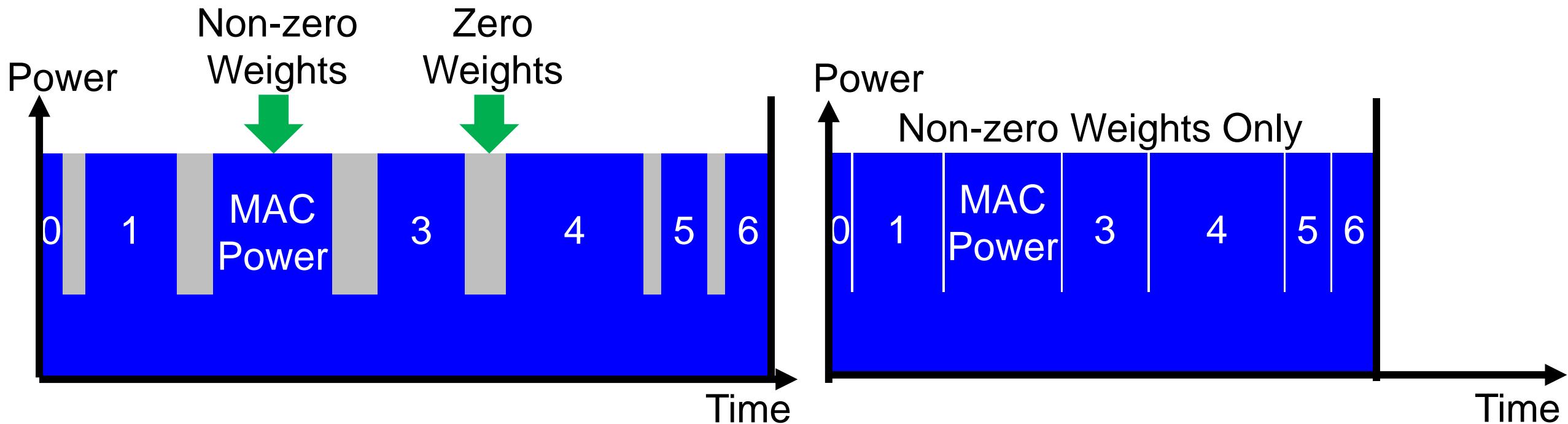
Cycle #3

Cycle #9

Skipping Convolution: Moving OFM



$$\text{Energy} = \sum \text{Power} \cdot \text{Time}$$

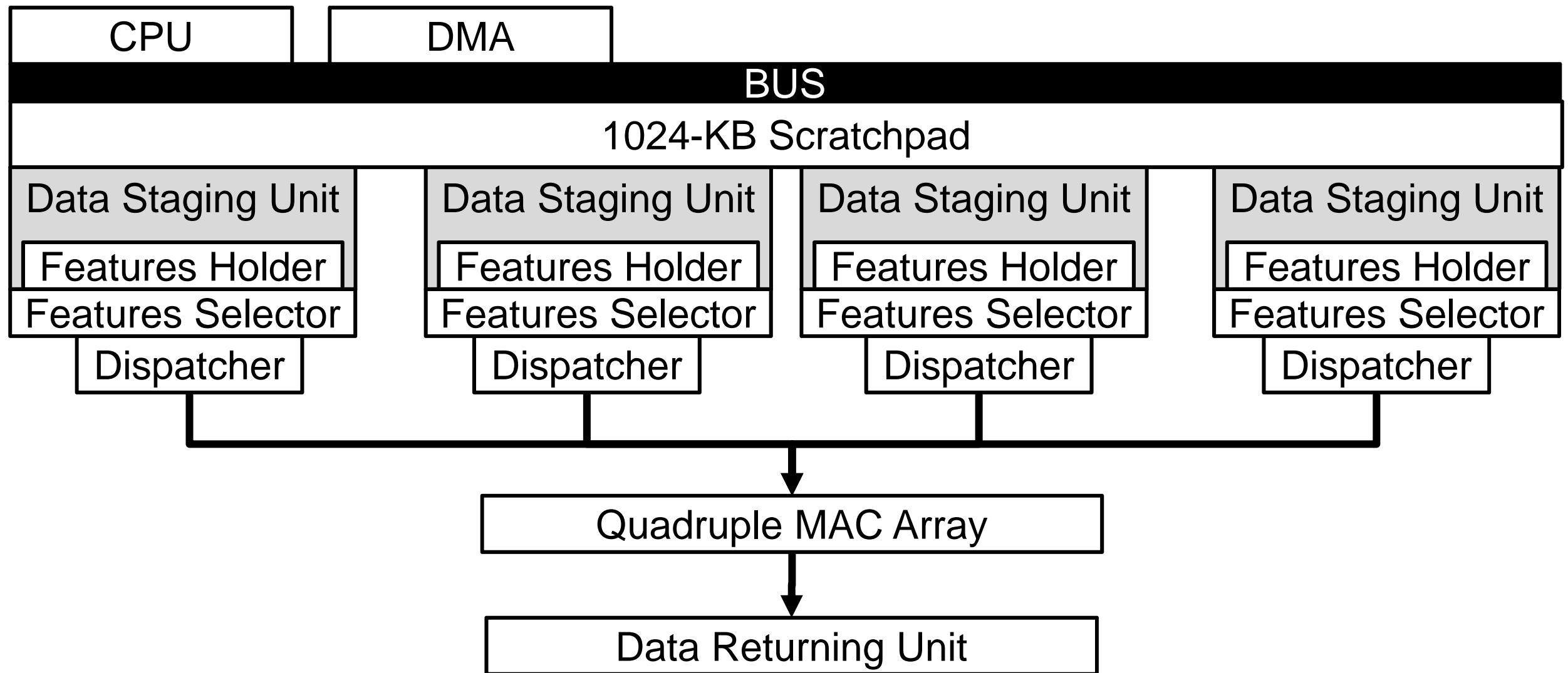


Energy of Non-skipping Convolution $>$ Energy of Skipping Convolution

Outline

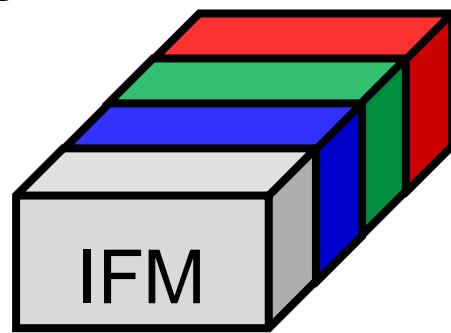
- Motivation
- Background
- Overall Architecture
- Key Features
 - Butterfly-Structure Dual-Core Accelerator
 - Sparsity-Aware Computing
 - Bandwidth-Efficient Network Traversal
- Measurement Results
- Conclusion

NPU: Overall Architecture

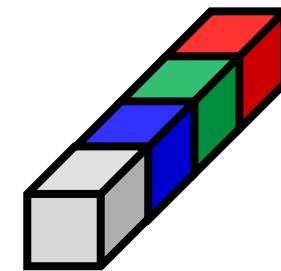


NPU: Convolution Layer

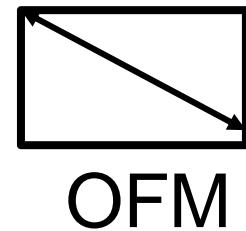
Input Channels
Width
Height



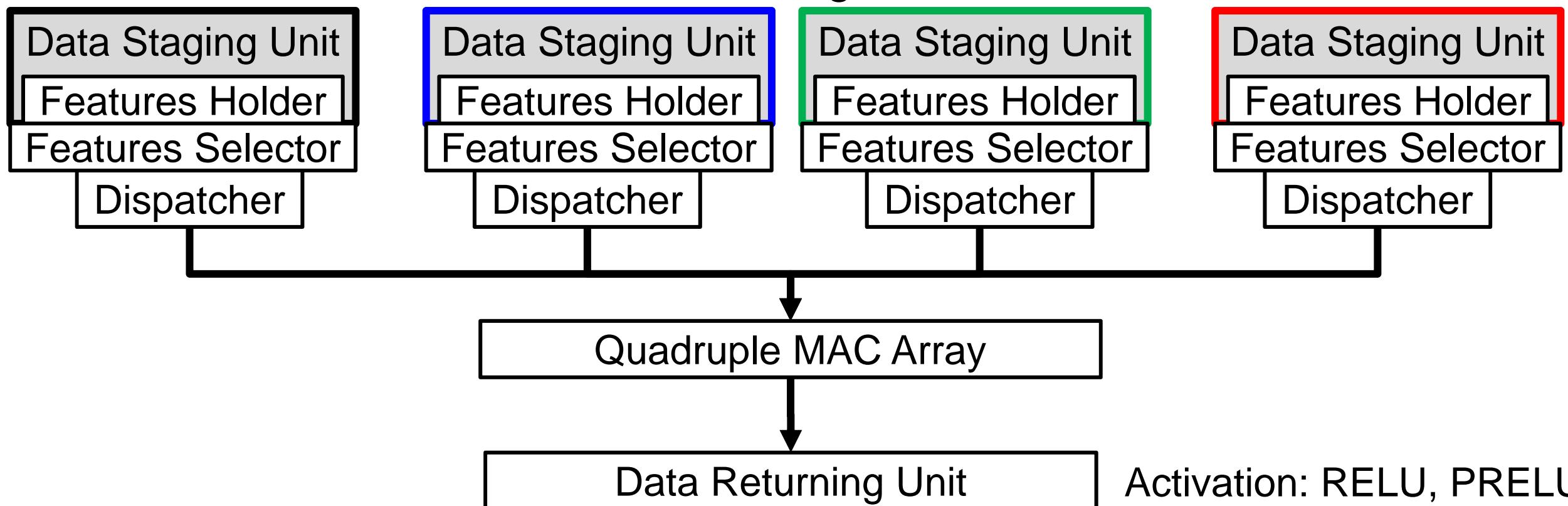
*



=



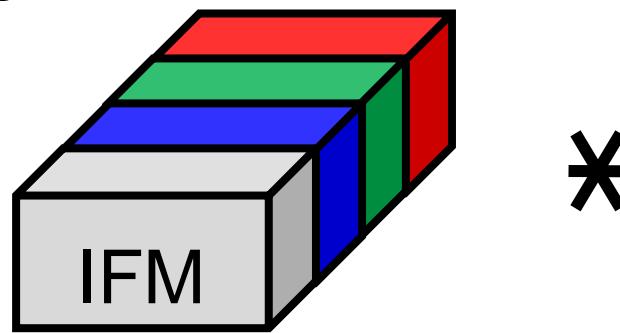
A Weight Kernel



Activation: RELU, PRELU

NPU: Convolution Layer

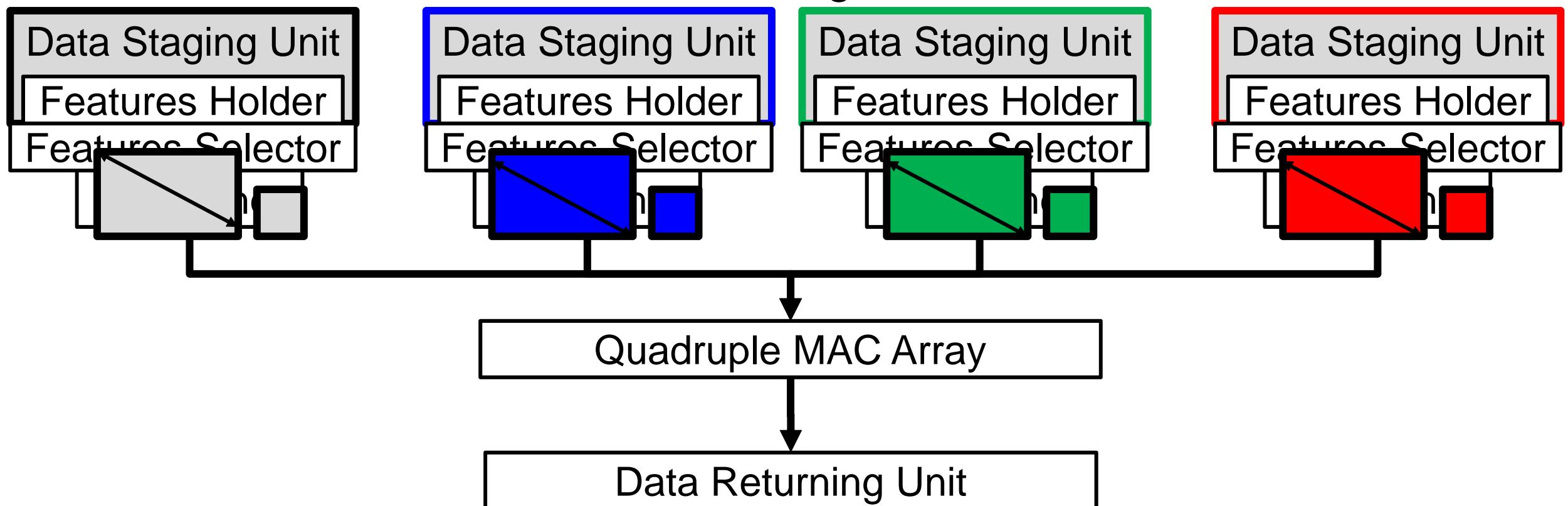
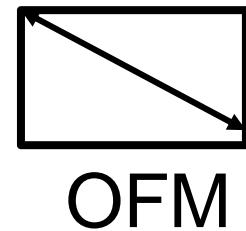
Input Channels
Width
Height



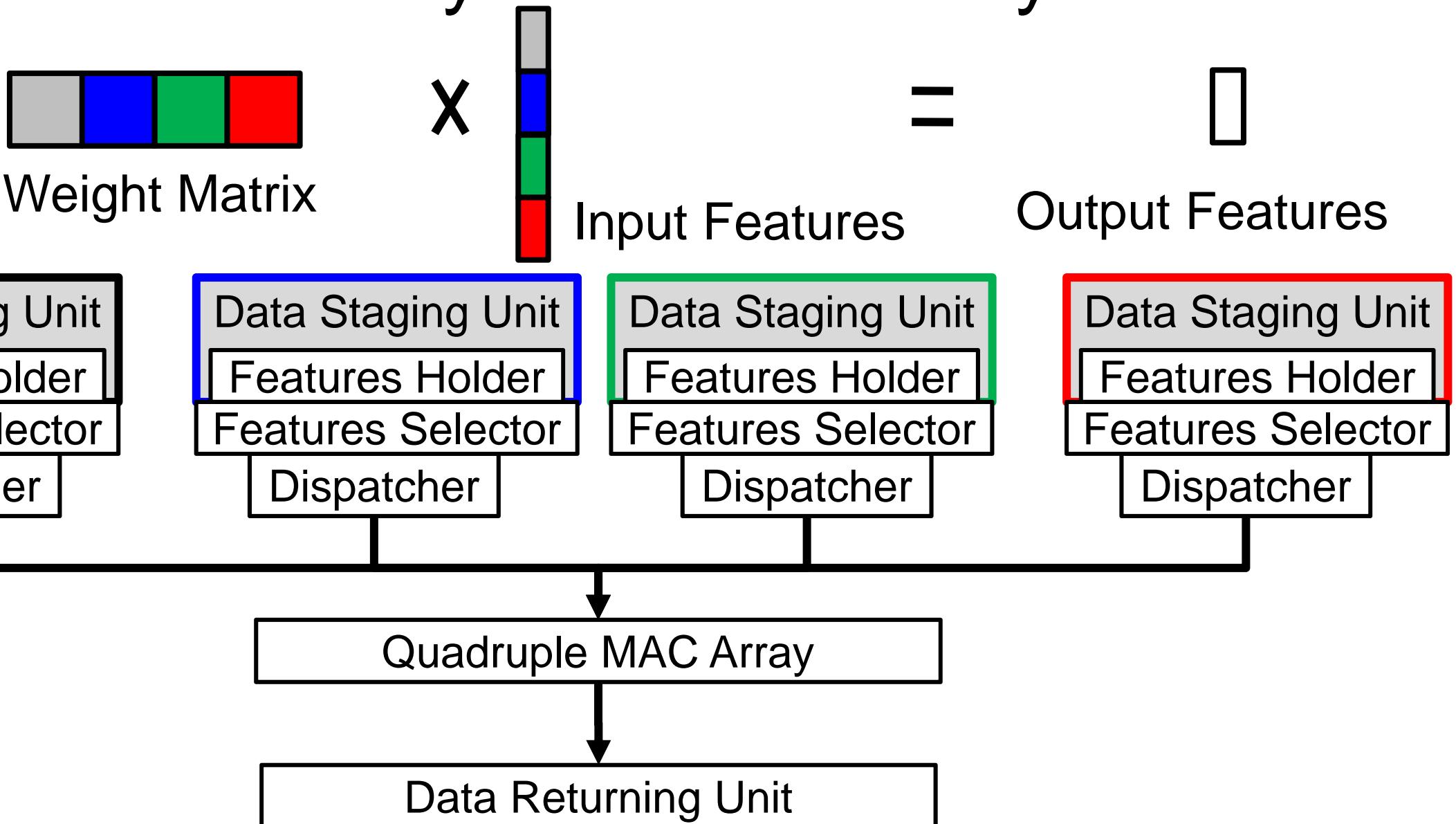
*

A Weight Kernel

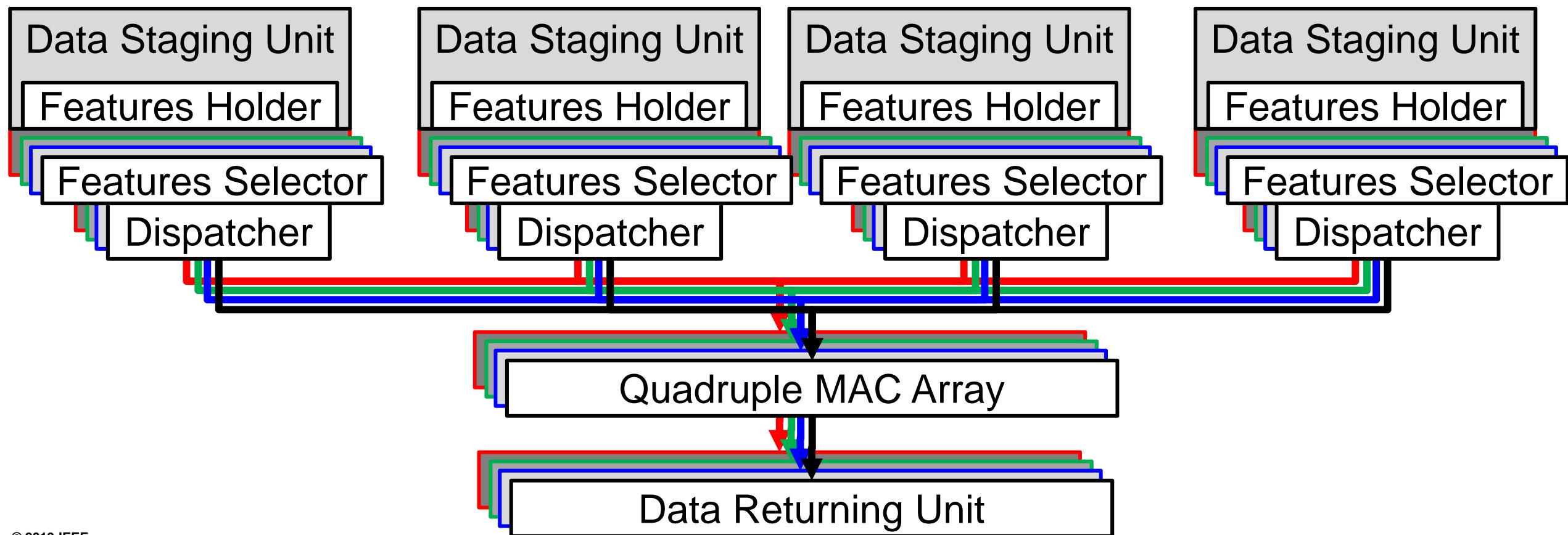
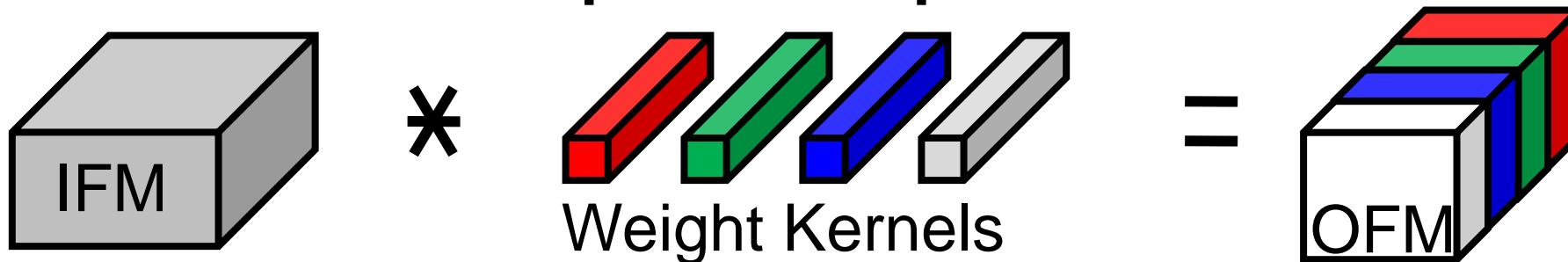
=



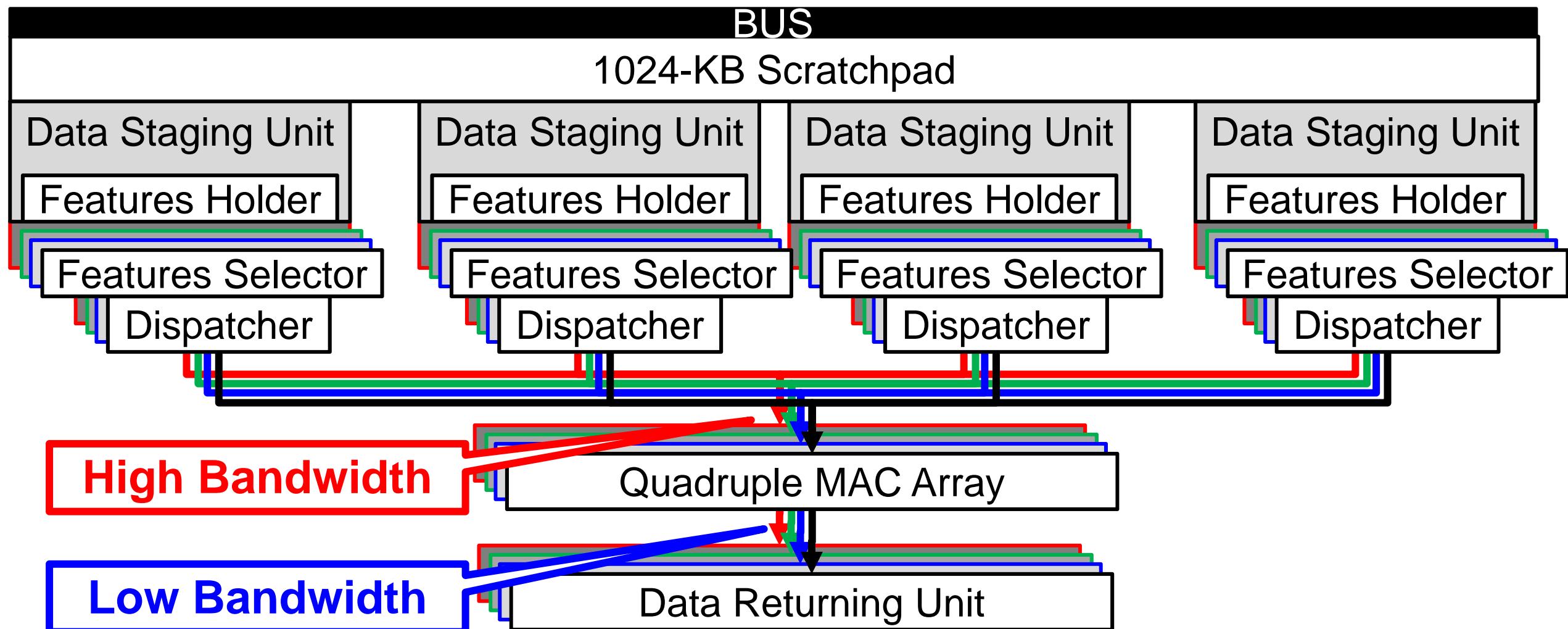
NPU: Fully-Connected Layer



NPU: Multiple Output Channels



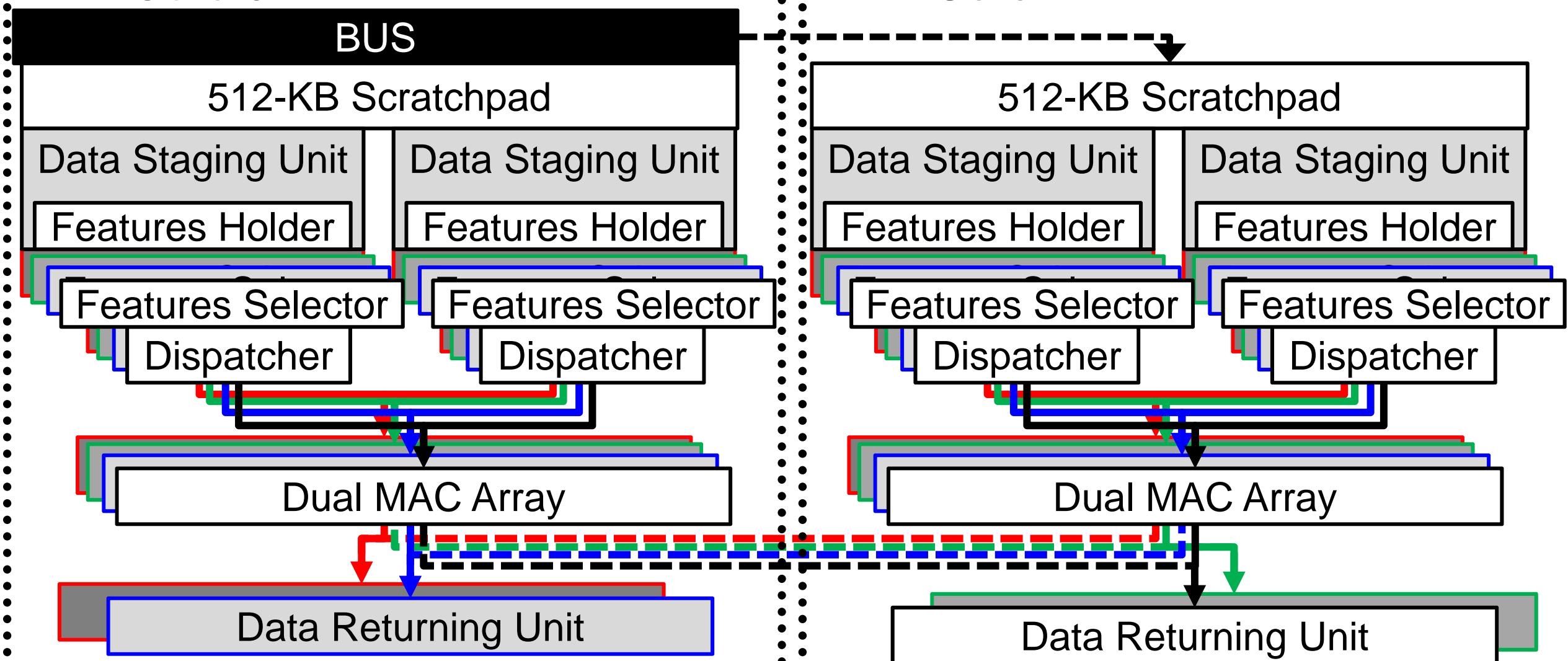
NPU: High Logic Cost and Wiring Congestion



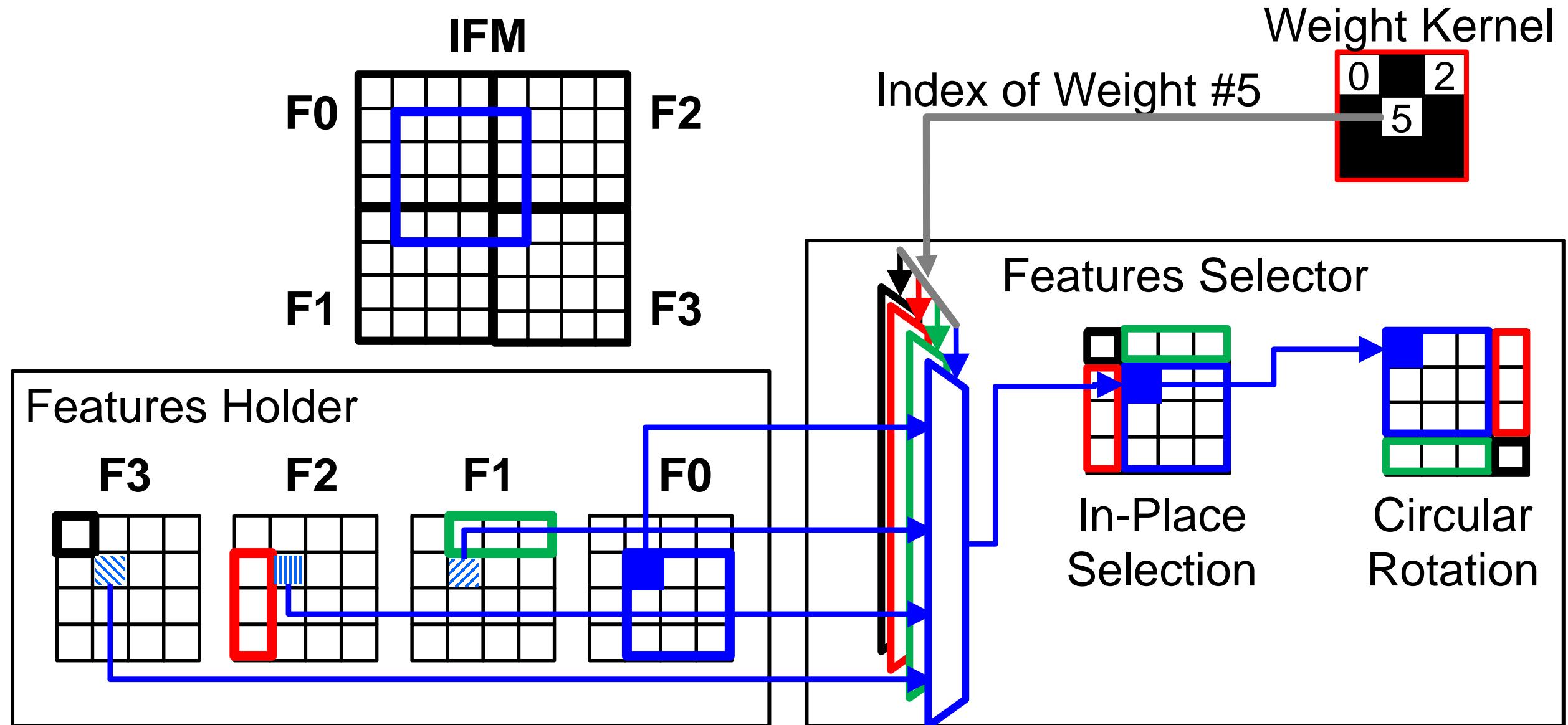
Butterfly-Structure Dual-Core Architecture

Core 0

Core 1

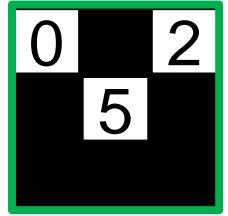


Sparsity-Aware Computing: Feature Selection

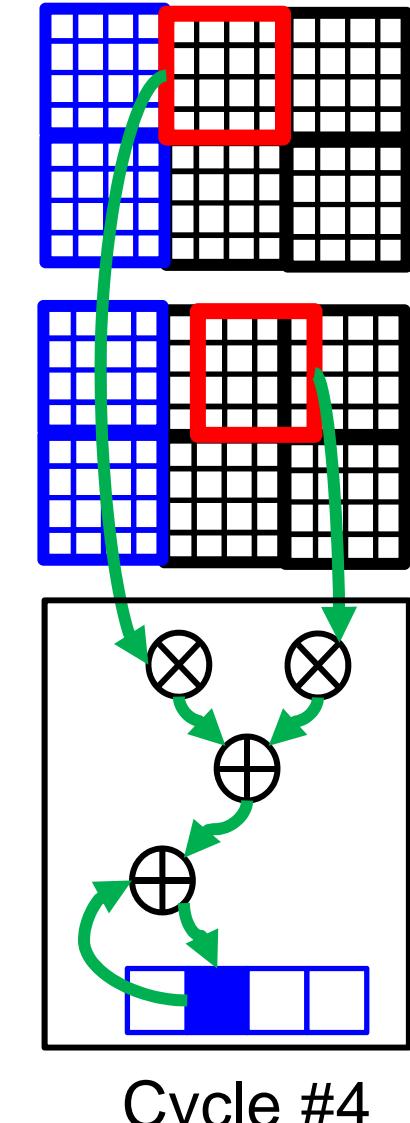
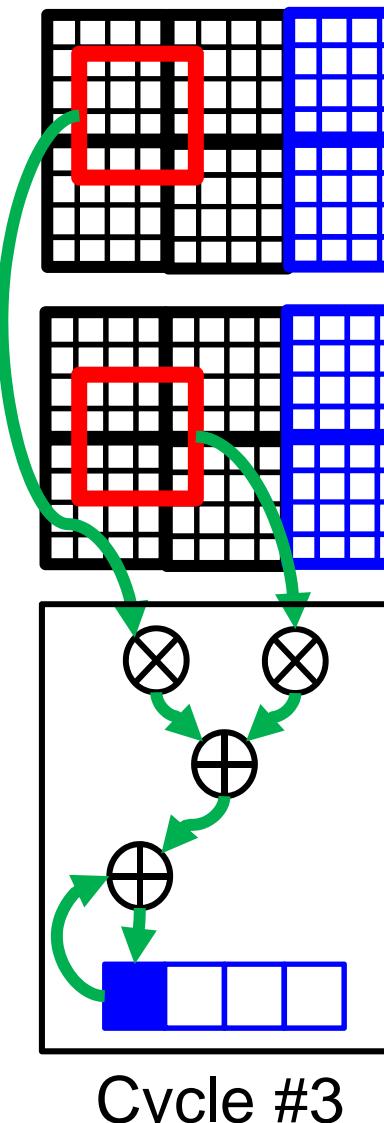
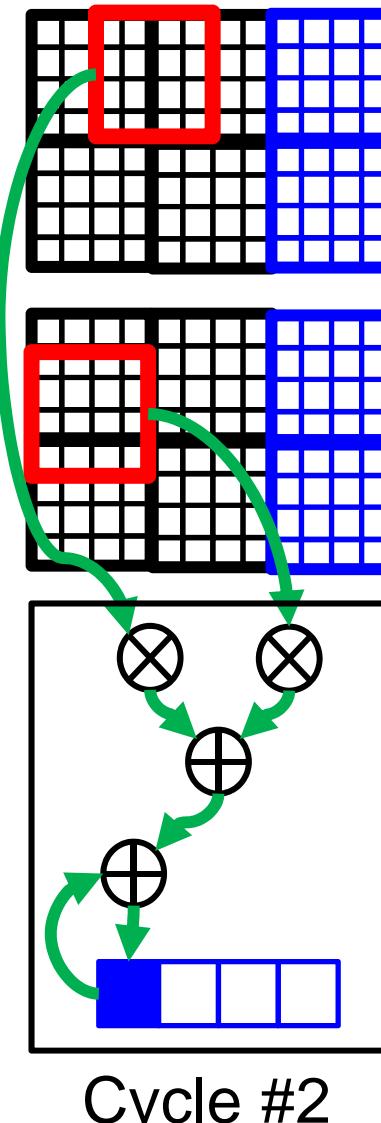
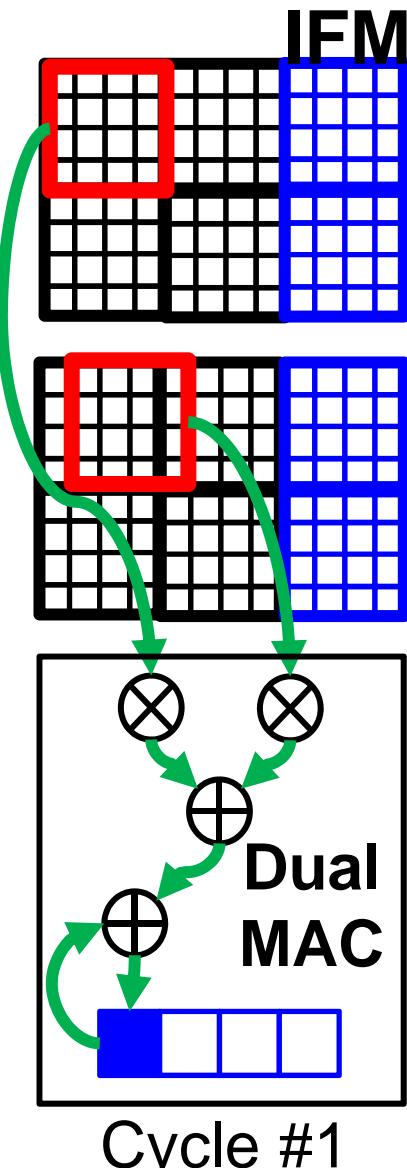
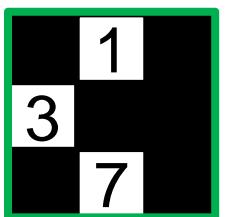


Sparsity-Aware Computing: Dual MAC Array

Weight Kernel
in DSU #0



Weight Kernel
in DSU #1



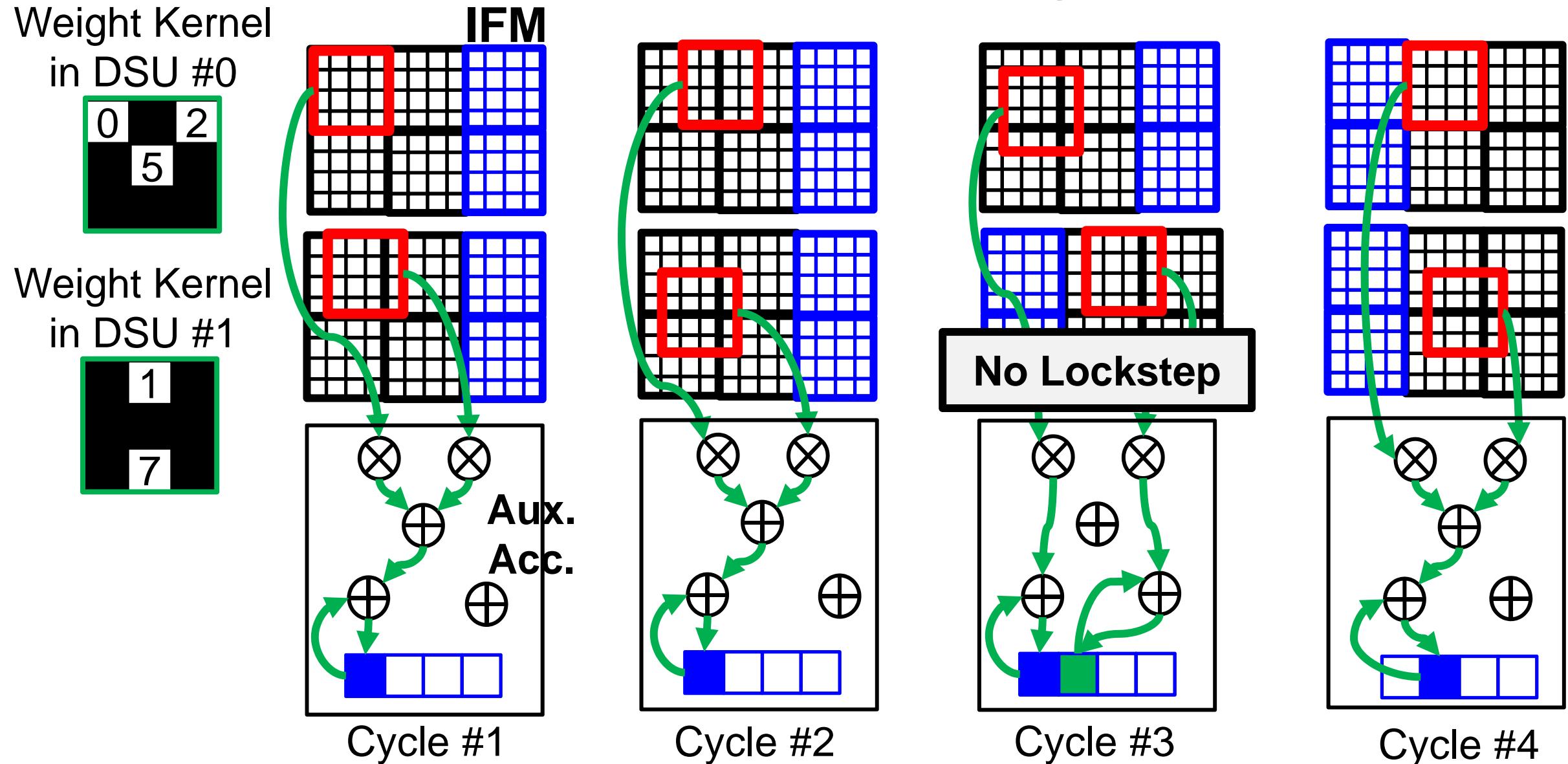
Cycle #1

Cycle #2

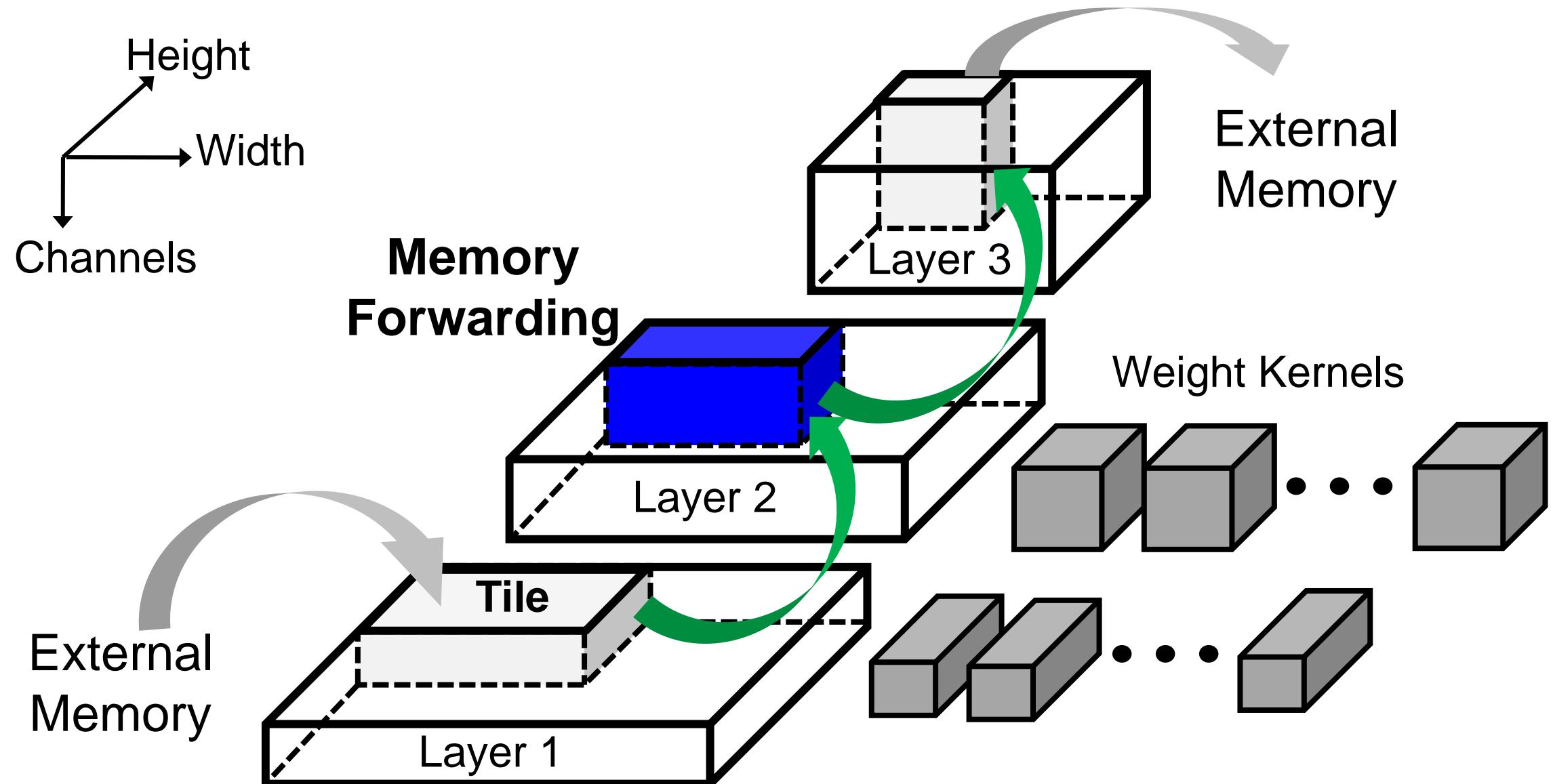
Cycle #3

Cycle #4

Sparsity-Aware Computing: Imbalance



Network Traversal: Feature-Map Forwarding

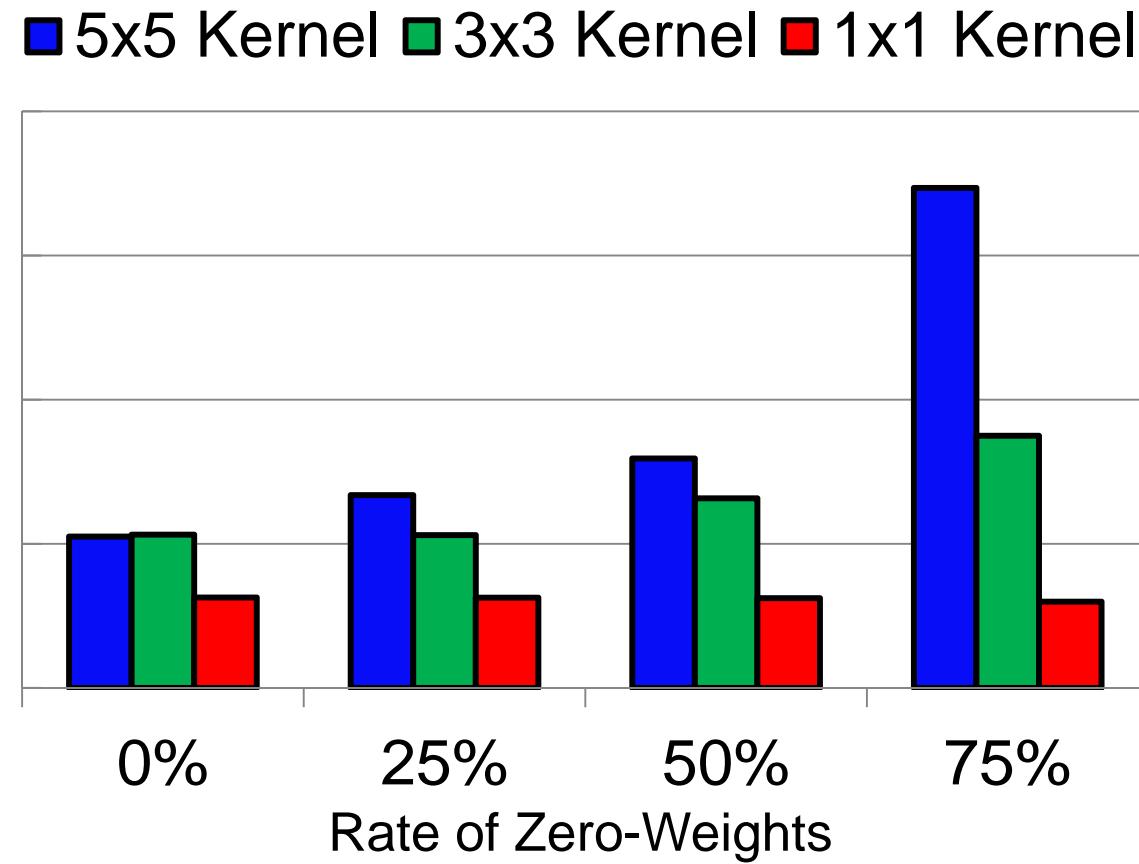


Outline

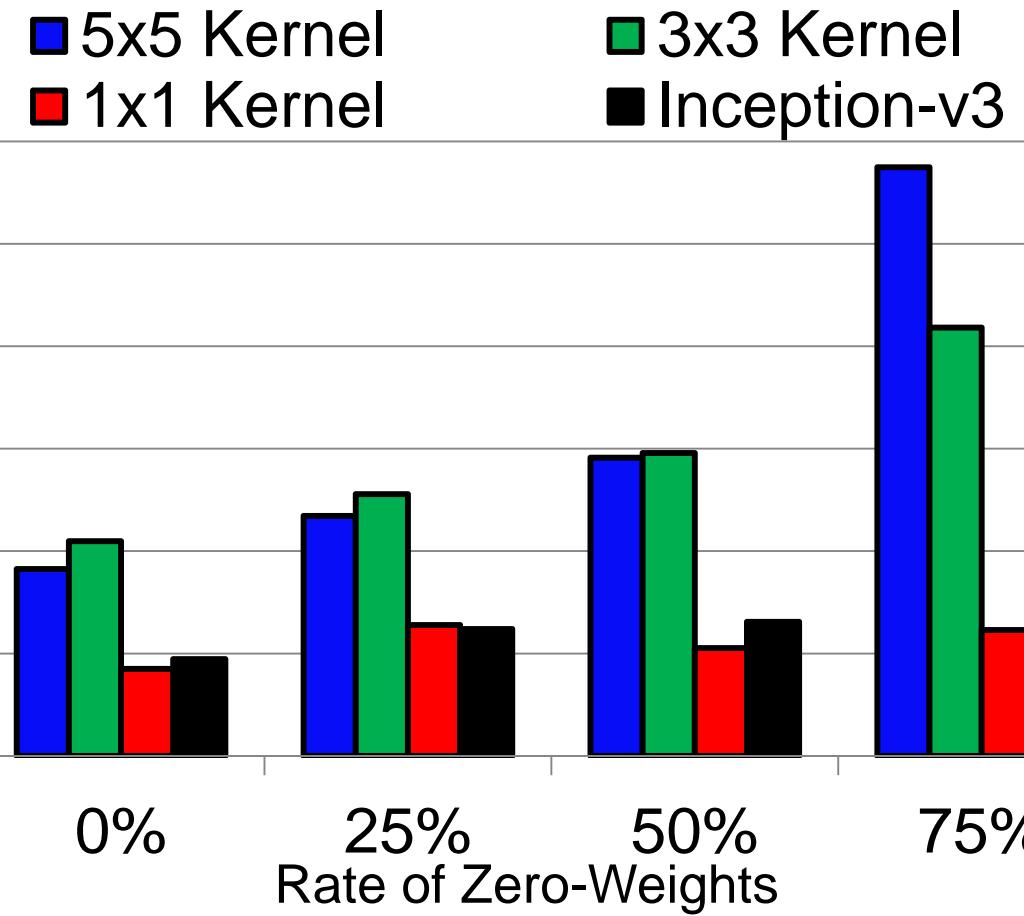
- Motivation
- Background
- Overall Architecture
- Key Features
 - Butterfly-Structure Dual-Core Accelerator
 - Sparsity-Aware Computing
 - Bandwidth-Efficient Network Traversal
- Measurement Results
- Conclusion

Measurement Results

Peak Performance
(TOPS @ 933 MHz)



Energy Efficiency
(TOPS/W @ 0.5V)



Comparison

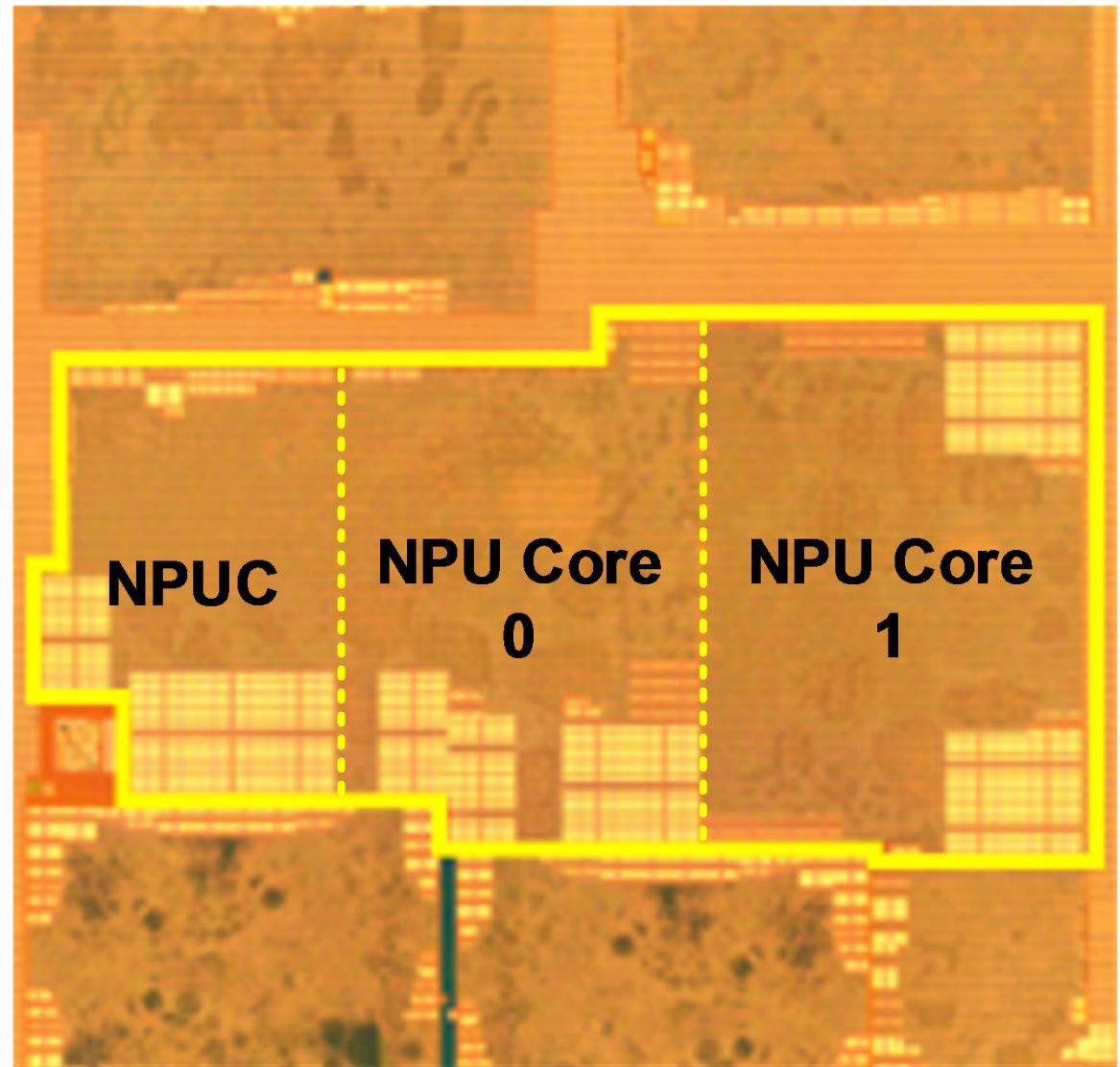
	This Work	Y. Chen, ISSCC 2016	G. Desoli, ISSCC 2017	J. Lee, ISSCC 2018
Process (nm)	8	65	28, FDSOI	65
Area (mm ²)	5.5	12.25	34	16
Voltage (V)	0.5 - 0.8	0.82-1.17	0.575-1.1	0.63-1.1
Frequency (MHz)	67 - 933	250	200 - 1175	2 - 200
On-Chip Memory (KB)	1568	181.5	5760	256
Bit Precision	8, 16	16	8, 16	1-16
Peak Performance (GOPS)	1910 (8b) 6937* (8b)	84	676 (8b)	7372 (1b), 691.2(8b)
Energy Efficiency (TOPS/W)	11.5* (8b) @ 0.5V 4.5* (8b) @ 0.8V	0.21	2.93 (8b) @ 0.575V	50.8 (1b), 5.57 (8b) @ 0.66V
Power (mW)	39 @ 0.5V 1,553 @ 0.8V	278 @ 200MHz	39 @ 0.575V	3.2 @ 0.63V 297 @ 1.1V

* 75% zero-weights

Die Photo

Process	Samsung 8nm
Area	5.5 mm ²
Voltage	0.5 V ~ 0.8V
Freq.	67 MHz ~ 933 MHz
Best Peak Performance	6.93* TOPS (8b) @ 0.8 V
Best Energy Efficiency	11.5 * TOPS/W (8b) @ 0.5 V

* 75% zero-weights



Conclusion

- Sparsity-aware computing for high performance and low energy
 - Streamlining the selection of input features without stall
 - Operating dual MAC together or independently for imbalanced nonzero weights
- Butterfly-structure architecture for mitigating logic costs and wiring congestion
 - Splitting the low-bandwidth channels behind the MAC arrays
- Feature-map forwarding between layers for saving memory bandwidth
 - Processing multiple layers within the scratchpad

A 20.5TOPS and 217.3GOPPS/mm² Multicore SoC with DNN Accelerator and Image Signal Processor Complying with ISO26262 for Automotive Applications

Yutaka Yamada, Toru Sano, Yasuki Tanabe, Yutaro Ishigaki,
Soichiro Hosoda, Fumihiko Hyuga, Akira Moriya, Ryuji Hada,
Atsushi Masuda, Masato Uchiyama, Tomohiro Koizumi,
Takanori Tamai, Nobuhiro Sato, Jun Tanabe, Katsuyuki Kimura,
Ryusuke Murakami, Takashi Yoshikawa

Toshiba Electronic Devices & Storage, Kawasaki, Japan

Outline

- **Background**
- **High Performance with Low Power Consumption**
- **Functional Safety**
- **Implementation Results**
- **Conclusion**

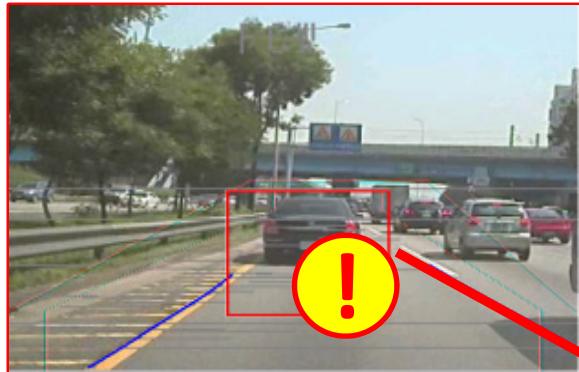
Outline

- **Background**
 - Target of the SoC
 - Our Previous Work
 - Requirements for the SoC
- High Performance with Low Power Consumption
- Functional Safety
- Implementation Results
- Conclusion

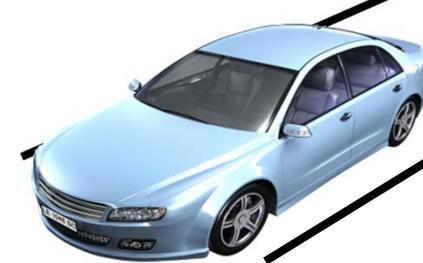
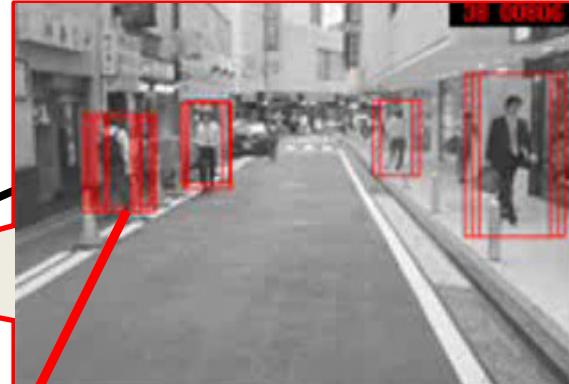
Target of the SoC

Advanced Driver Assistance Systems (ADAS)

Vehicle detection



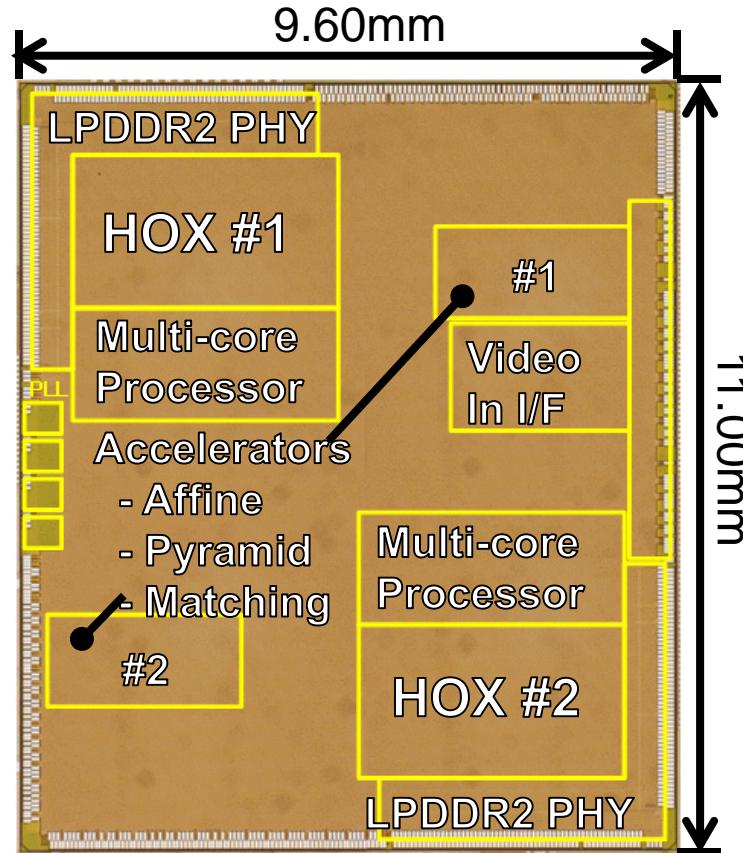
Pedestrian detection



Autonomous Emergency
Braking (AEB)

Several image recognition applications run in ADAS

Our Previous Work [ISSCC 2015]



Process	40nm LP
Chip Size	105.6mm ²

Concept

- High performance with low power consumption
 - 1.9TOPS, 3.37W, 564GOPS/W
 - Handles HD resolution

Approach

- Heterogeneous architecture
 - 2 clusters of 4-core RISC processors
 - 8 types of hardware accelerators

Requirements for the SoC

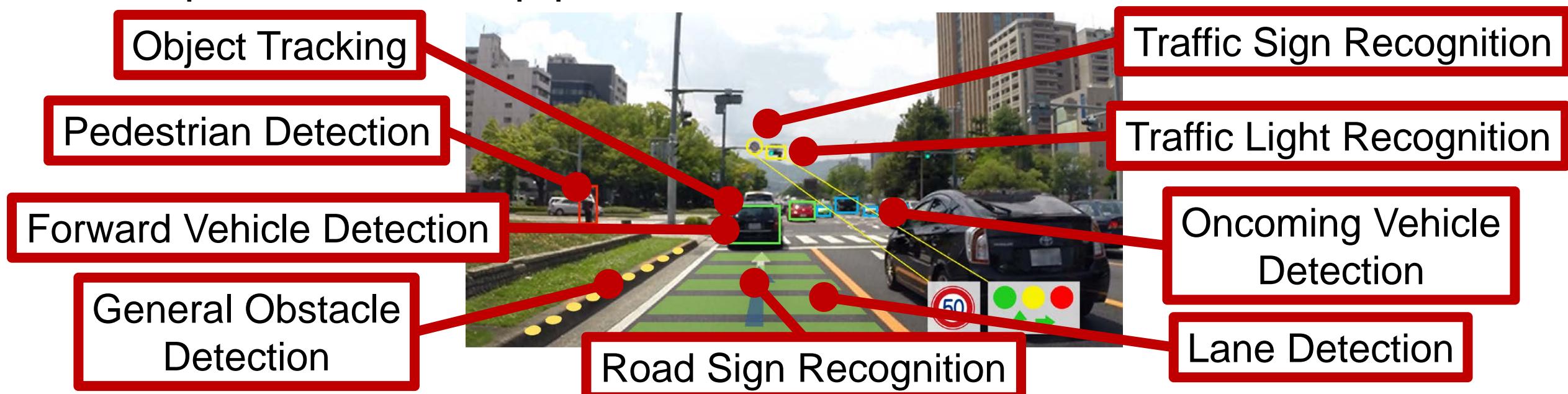
- #1: High performance with low power consumption
- #2: Functional safety

Outline

- Background
- **High Performance with Low Power Consumption**
 - Requirements & Approaches
 - Architecture of the SoC
 - Hardware Accelerators
 - Performance
- Functional Safety
- Implementation Results
- Conclusion

Requirement #1 : High Performance with Low Power Consumption

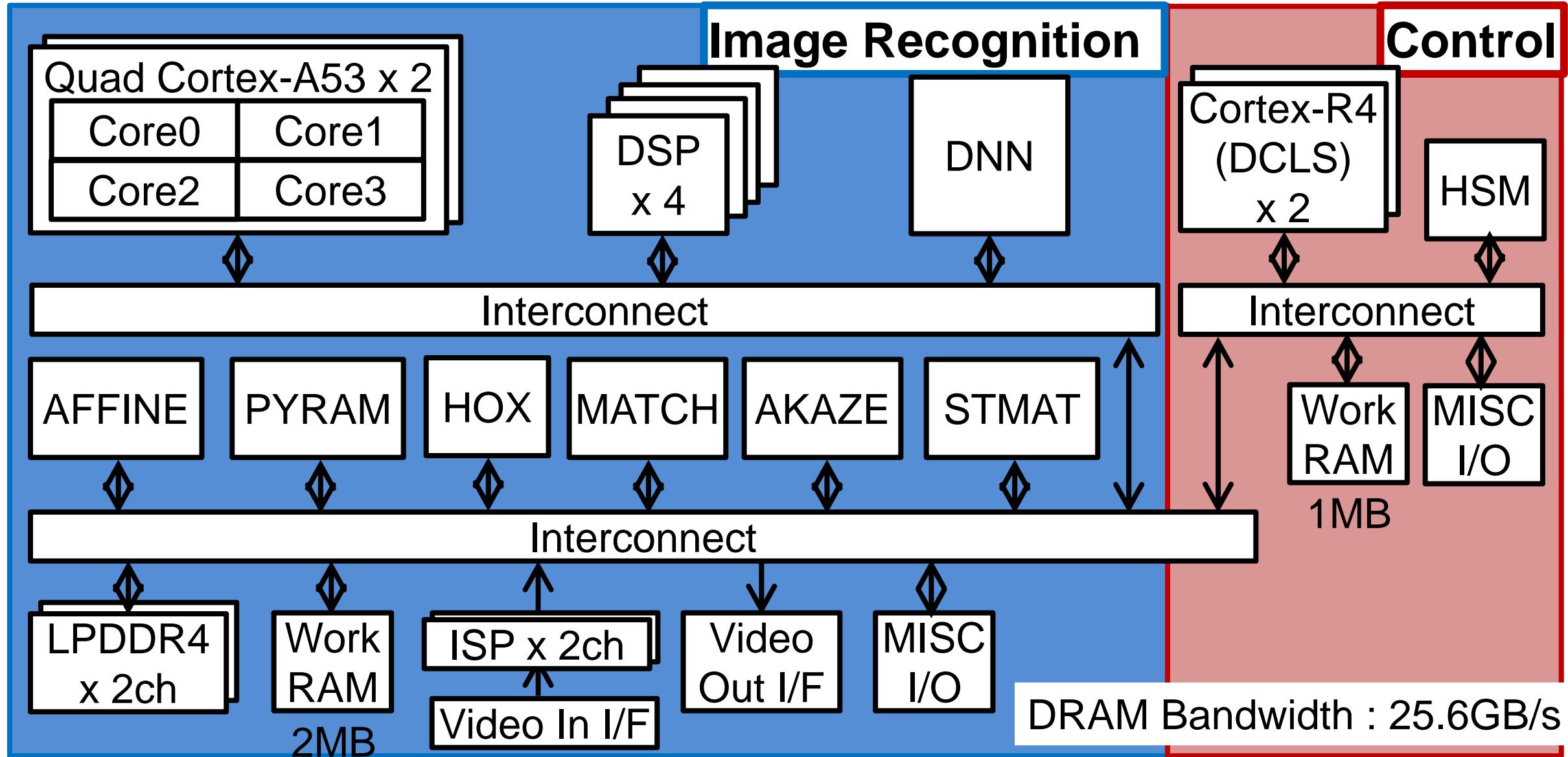
- SoCs for ADAS perform many image recognition applications
- To improve safety, more computing power is needed
 - More accurate, more object detection & tracking, and more detail
- Low power need keep performance stable in vehicle



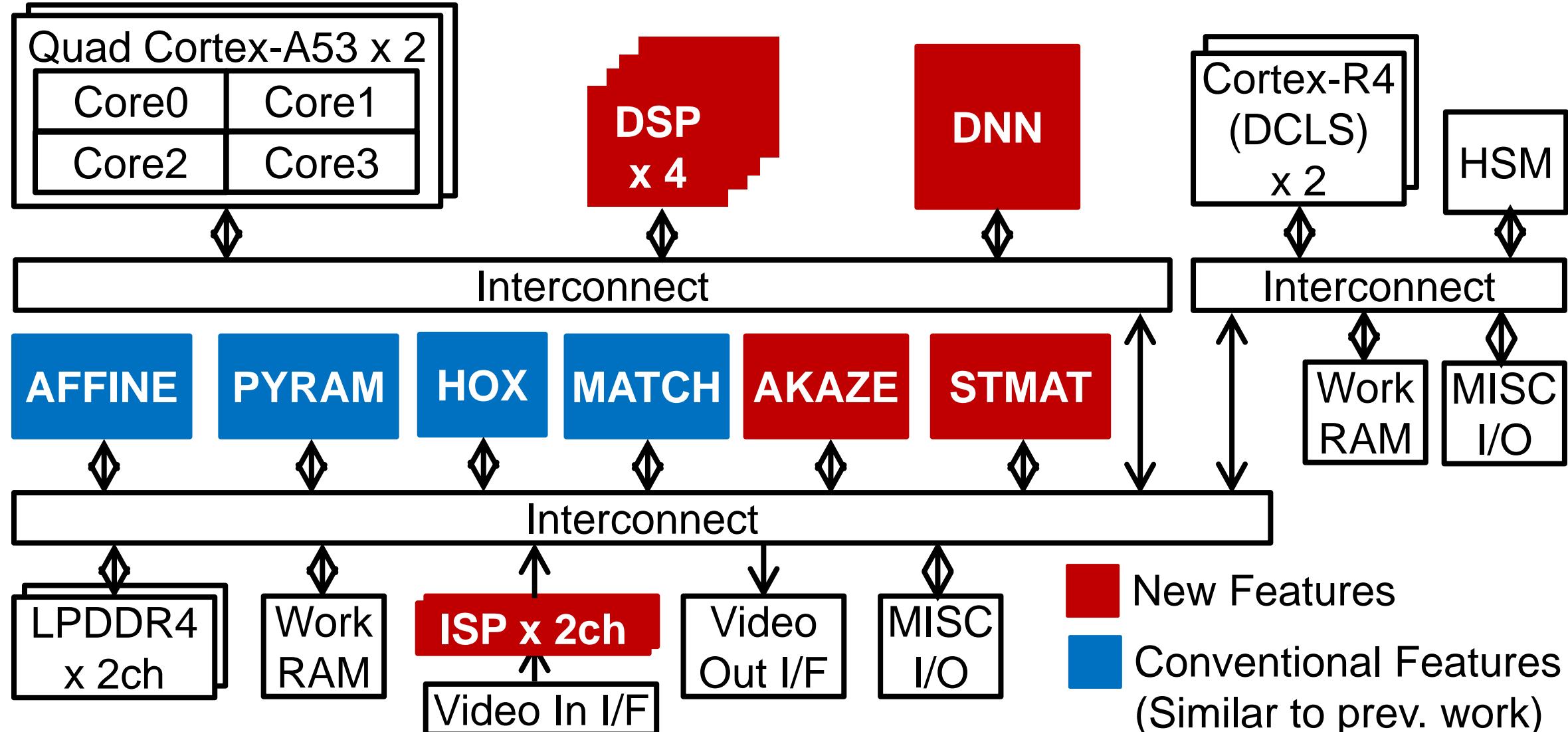
Approaches for High Performance with Low Power Consumption

- Heterogeneous architecture
 - Multi processors & multi DSPs
 - For applications with complexity and/or flexibility.
 - Power-aware hardware accelerators (HWAs)
 - For image recognition applications with high parallelism
- The approaches of HWAs for power efficiency
 - Parallel architecture for frequency reduction
 - DRAM access reduction

Architecture of the SoC



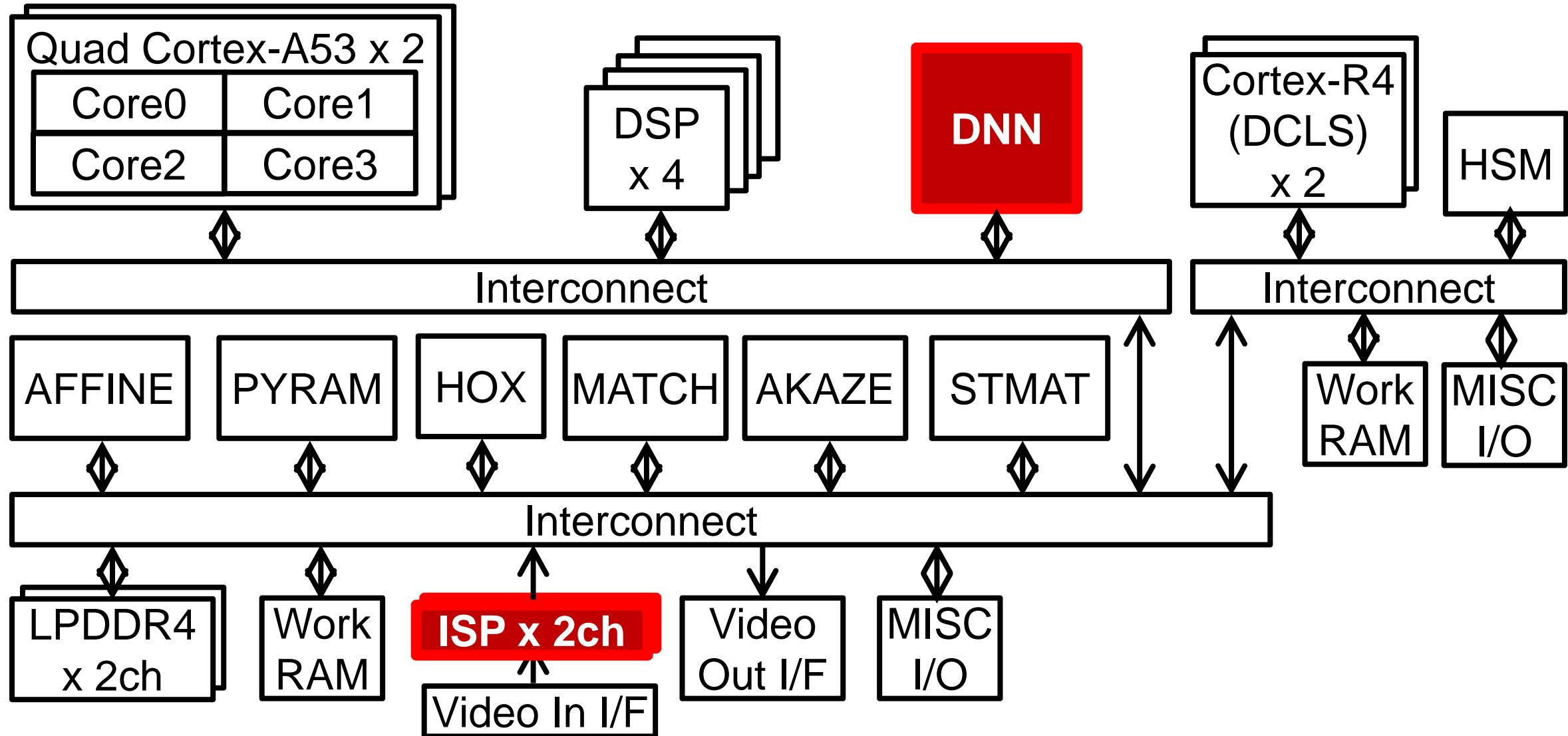
Features Updated from Our Previous Work



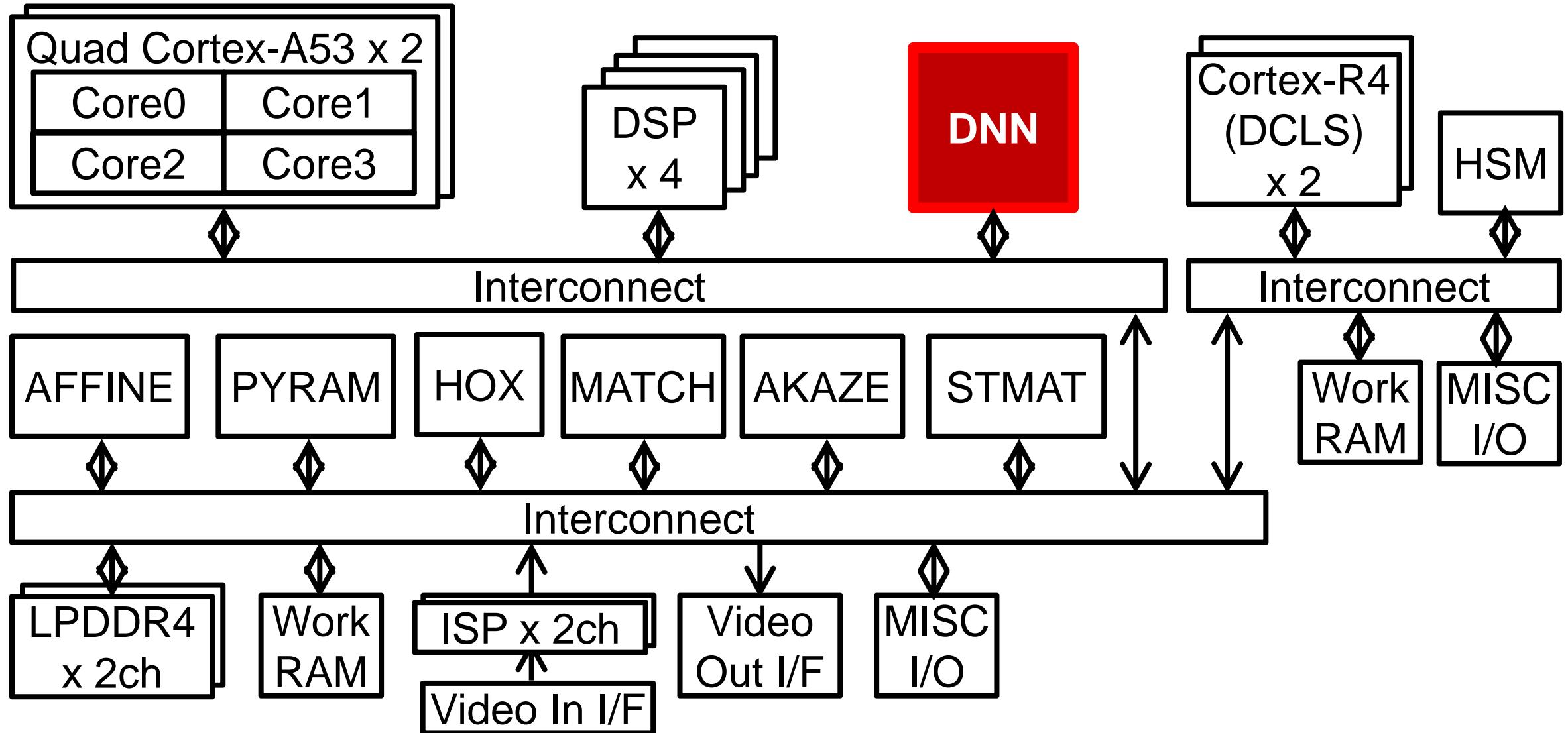
Hardware Accelerators to Improve Safety

- More accurate / More object detection & tracking
 - Introducing advanced algorithm for detection & tracking
 - Increase computing power
- **Accelerators with advanced algorithm**
 - DNN for object detection & semantic segmentation
 - STMAT for depth map generation
 - AKAZE for tracking a lot of objects
- More detail
 - Support high resolution and high dynamic range
- **Image Signal Processor (ISP)**

Hardware Accelerators (HWAs)



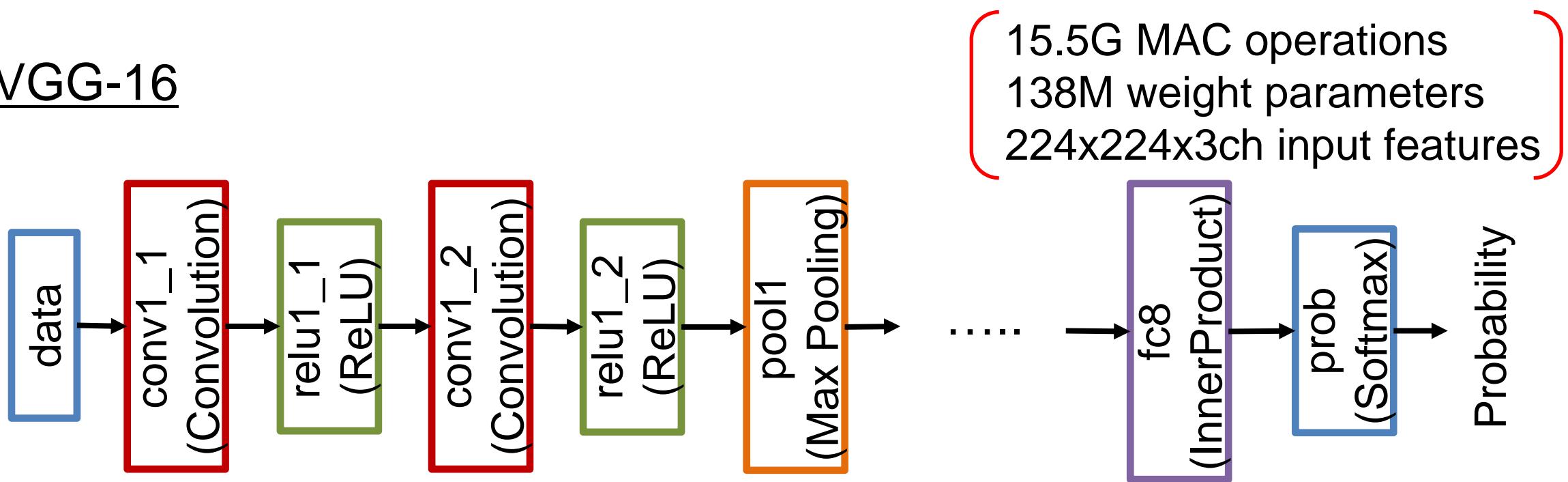
DNN Accelerator



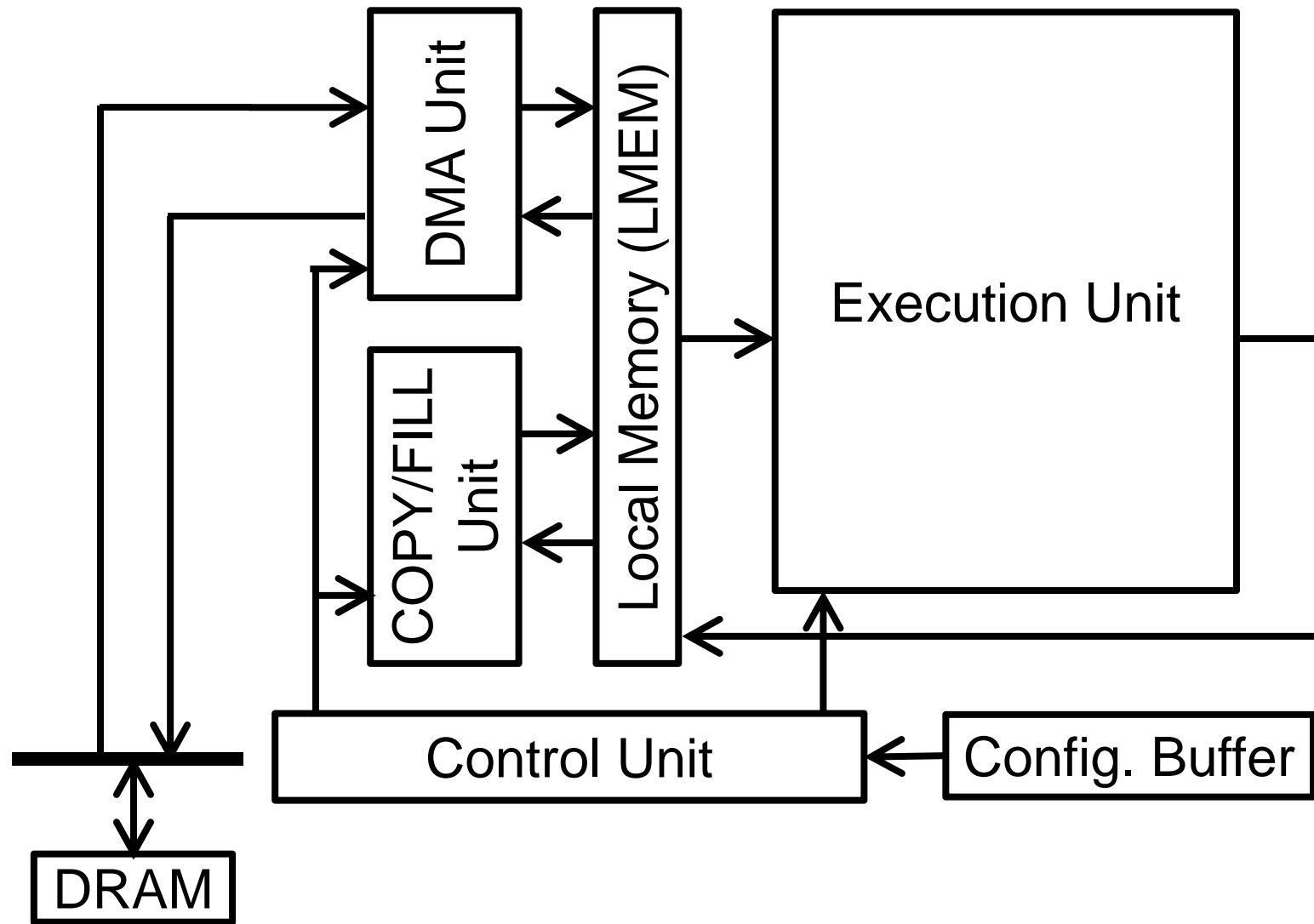
Deep Neural Network (DNN)

- One technology of machine learning
 - High accuracy for Computer Vision
 - Operations require a lot of MAC calculations and large bandwidth

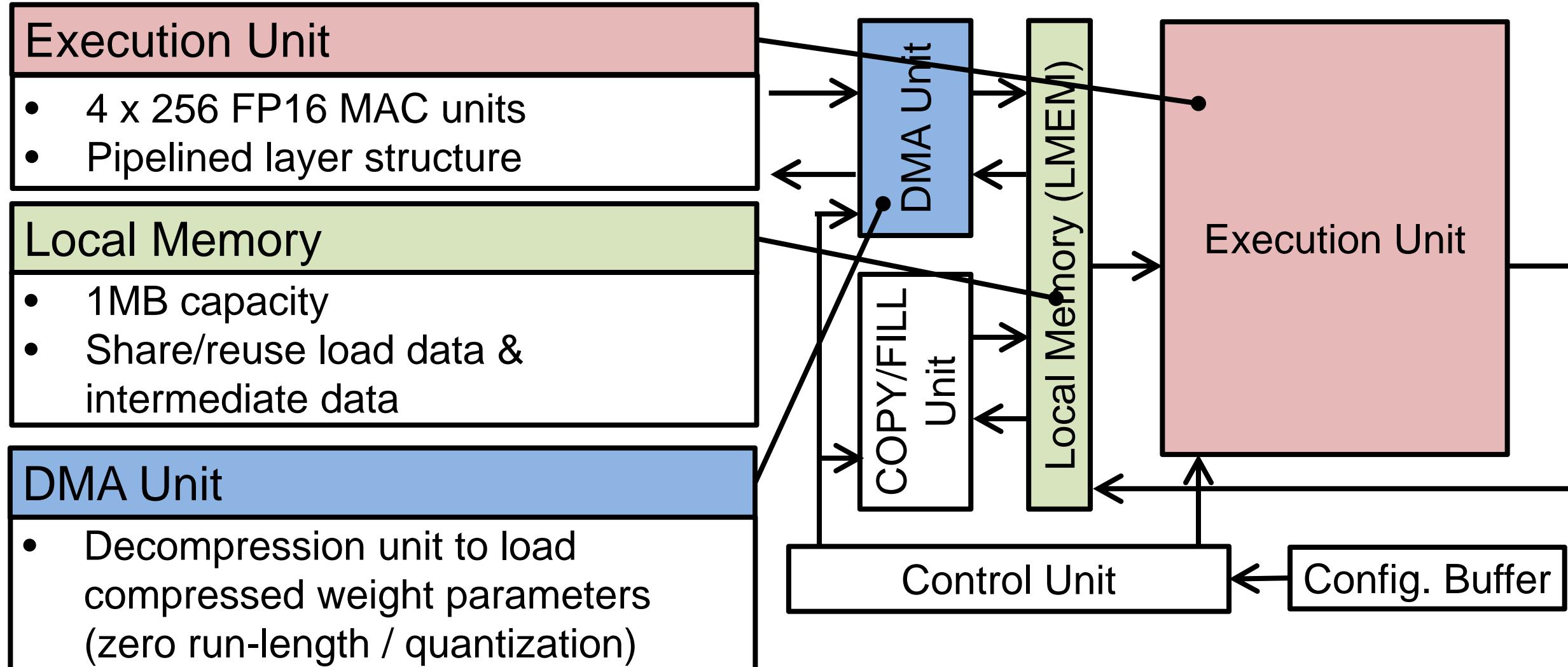
VGG-16



Architecture of DNN Accelerator



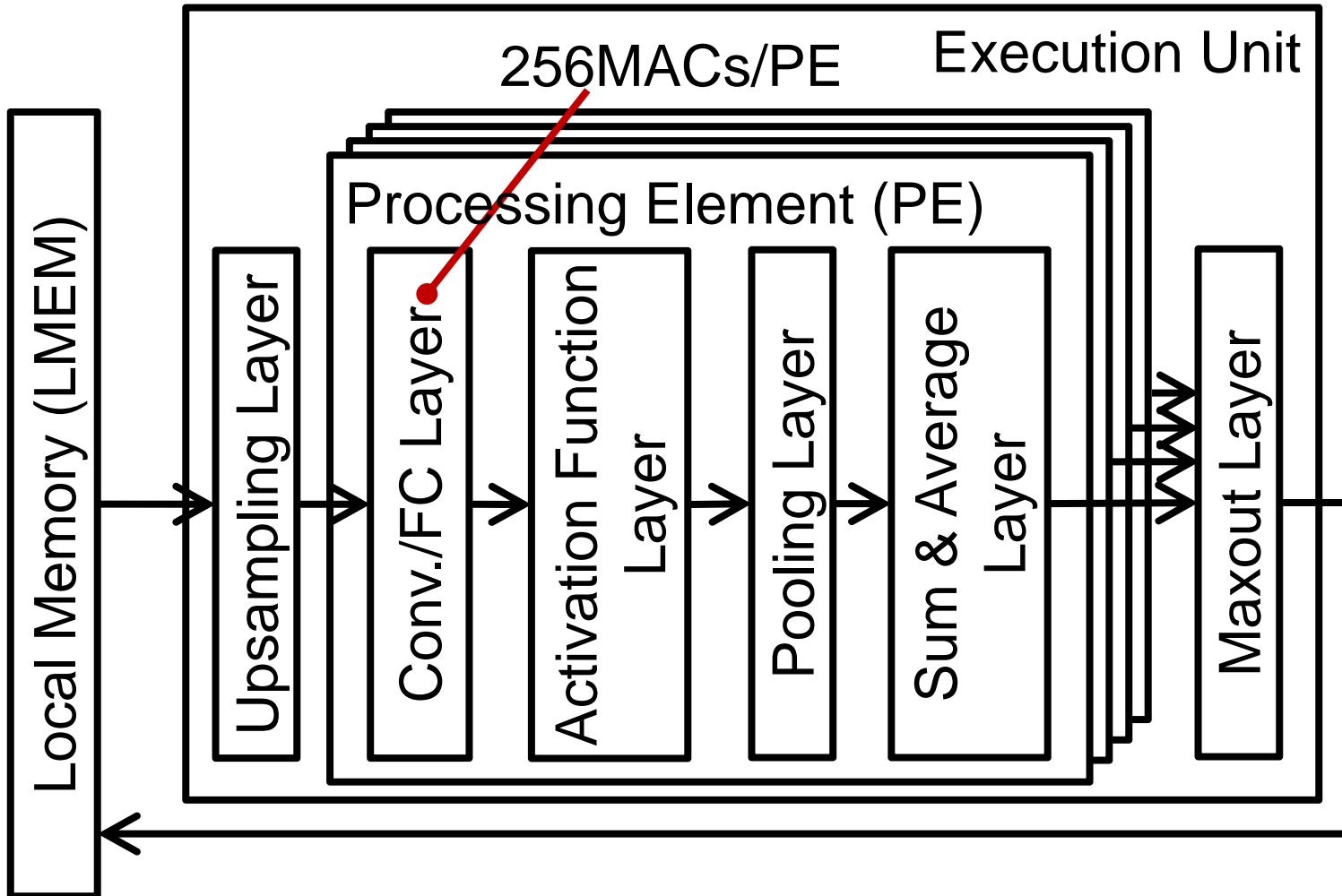
Features for Low Power Consumption



Structure of Execution Unit

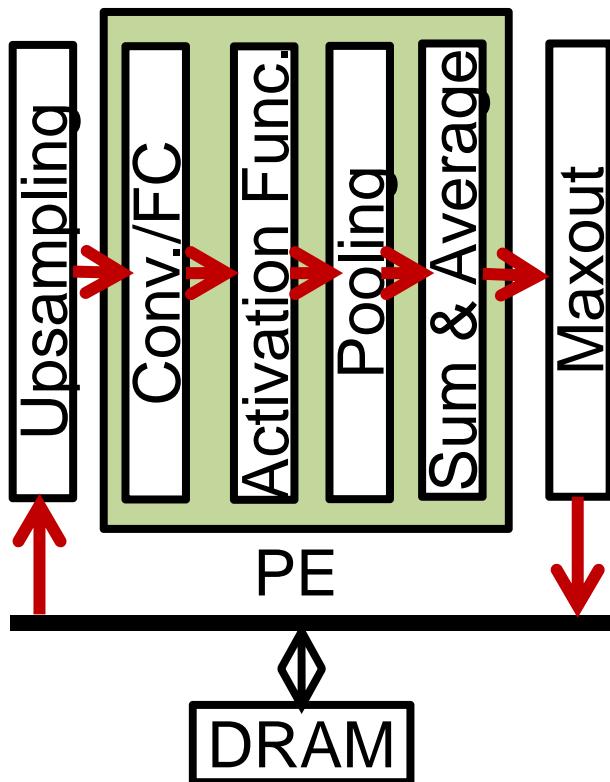
Pipelined layer structure can perform a series of DNN operations **in one read and write local memory access**

4 PEs share same input feature maps and simultaneously compute different channels for output feature maps

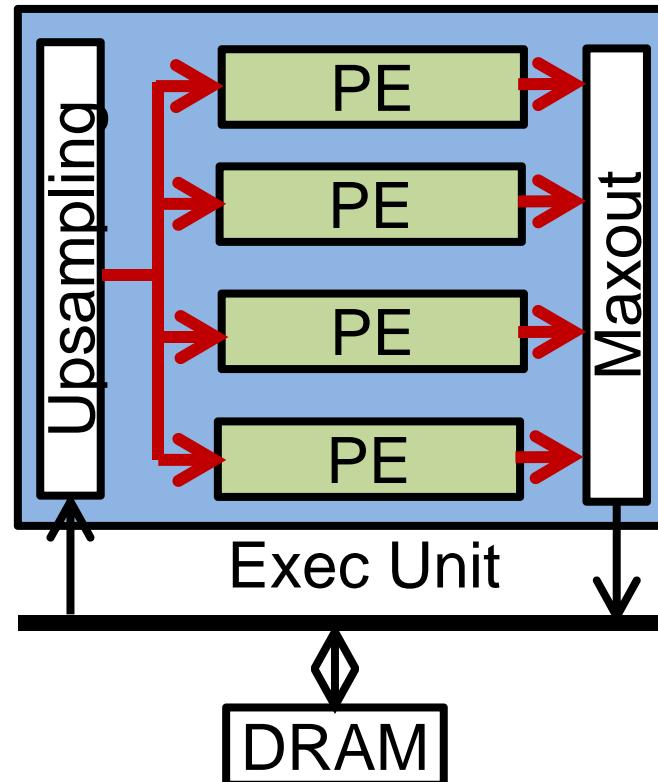


DRAM Bandwidth Reduction

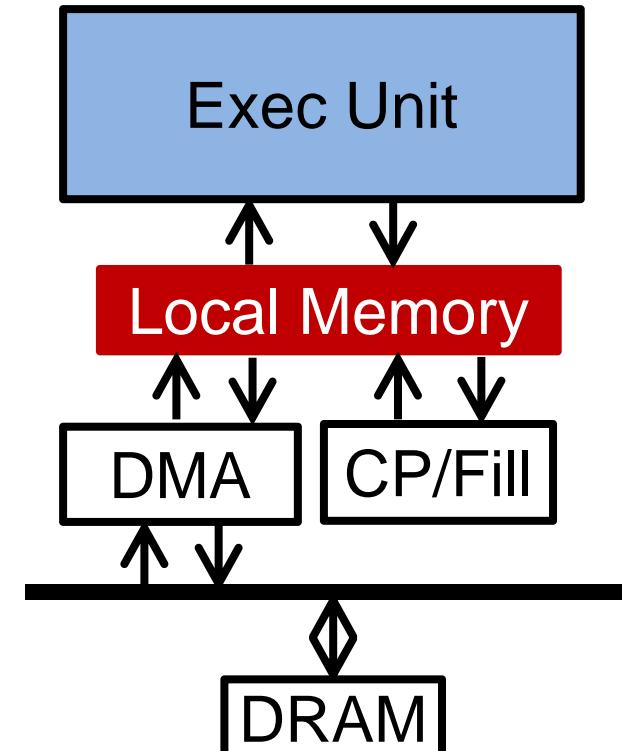
Pipelined Structure
(w/o Local Memory)
48GB/s



Input Feature Map Sharing
(w/o Local Memory)
17GB/s



Proposed
Local Memory
w/ Optimization by Tool
3.7GB/s



Use case : VGG-16

Image Signal Processor (ISP)

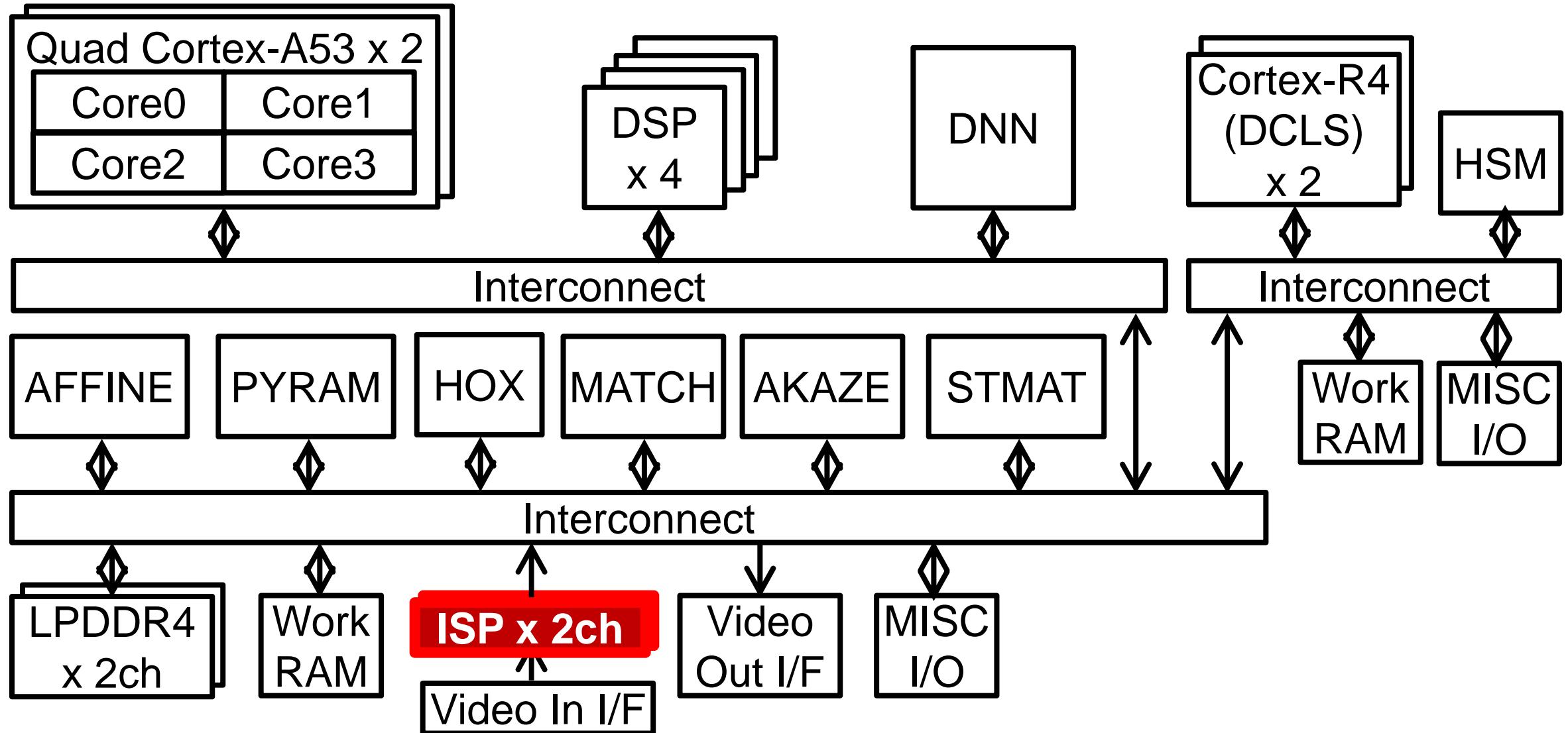
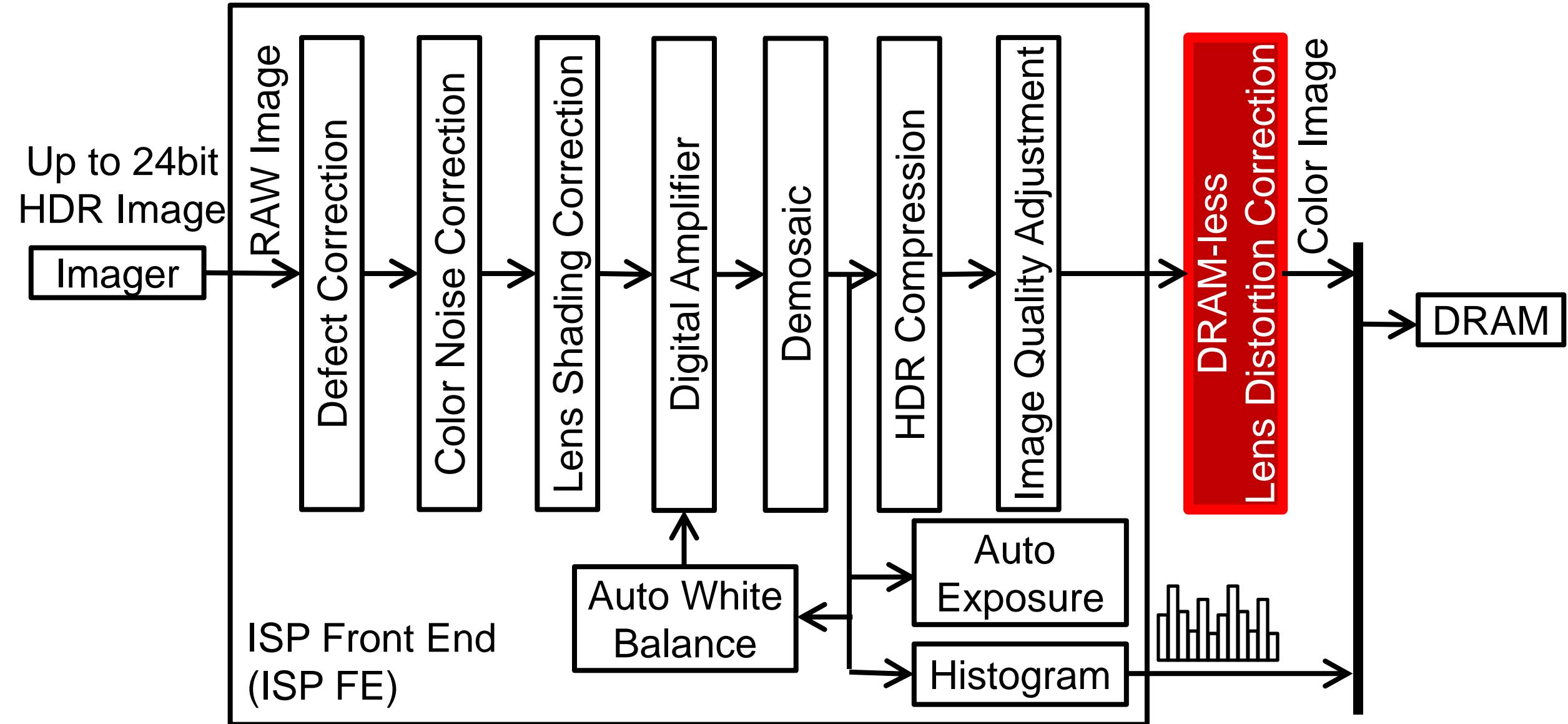


Image Signal Processor (ISP)



Lens Distortion Correction (LDC) Unit

Conventional (w/ DRAM)

LDC cannot connect ISP FE directly

Tiling



LDC



Tiling



Proposed (w/o DRAM)

LDC can connect ISP FE directly

Raster



LDC



Random

Includes
Larger Buffers



Lens Distortion Correction (LDC) Unit

Conventional (w/ DRAM)

LDC cannot connect ISP FE directly

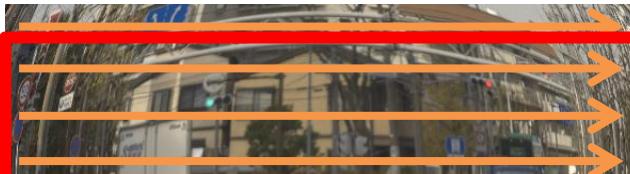
Tiling



Proposed (w/o DRAM)

LDC can connect ISP FE directly

Raster



**Proposed LDC need not load input image from DRAM
and can reduce bandwidth by **5.14GB/s**
(Use Case: ISP 2 ch. / 2880 x 1860 RGB color image @ 40fps)**

Tiling



Random



Larger Buffers

Performance Evaluation

	Performance [GOPS]	Power [mW]	Efficiency [GOPS/W]
TOTAL	20,537	9,776	2,100.8
ARM CA53	256	1,286	199.1
DSP	1,024	1,377	743.6
DNN	1,597	1,593	1,002.8
ISP	8,880	1,579	5,623.8
STMAT	3,480	419	8,305.5
AKAZE	2,681	726	3,692.6
MATCH	1,175	254	4,627.6
PYRAM	396	130	3,046.2
AFFINE	204	281	726.0
HOX	842	1,696	496.7
ARM CR4	1	435	2.8

Condition

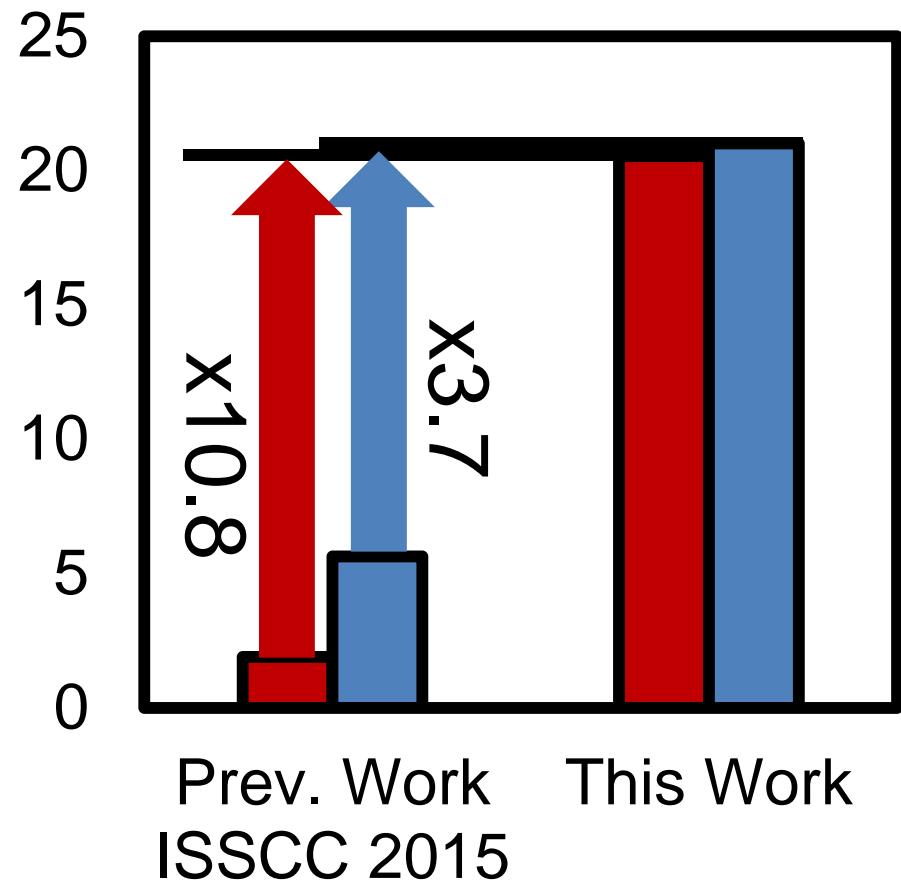
- Vdd=0.8V
- Process=center
- Temp=25° C

Note: 1 operation of OPS means 1 arithmetic or logical operation

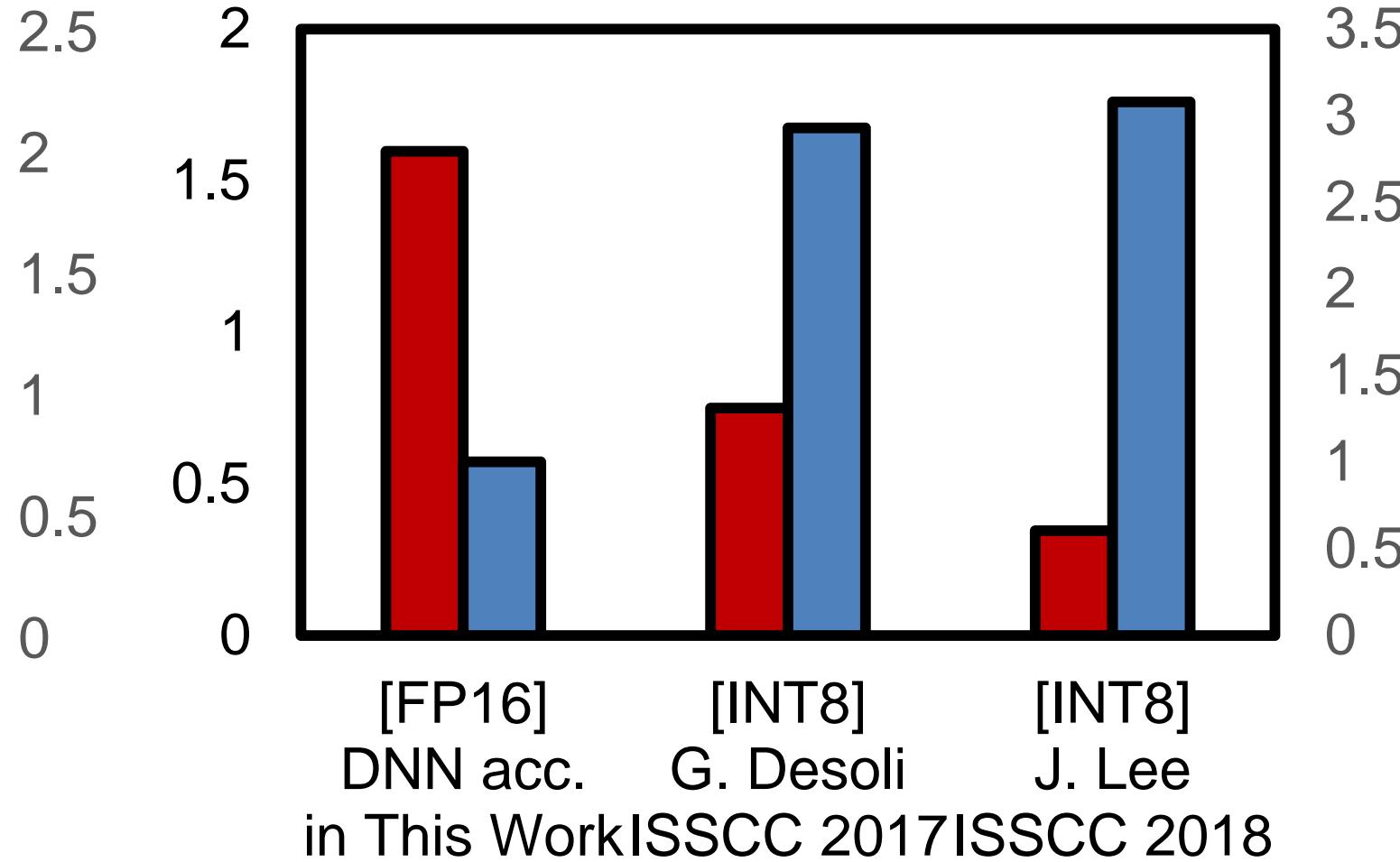
e.g. 1 MAC operation is 2 operations (MUL + ADD)

Performance Comparison

Total Performance



DNN Performance



■ Performance (Left Axis) [TOPS]

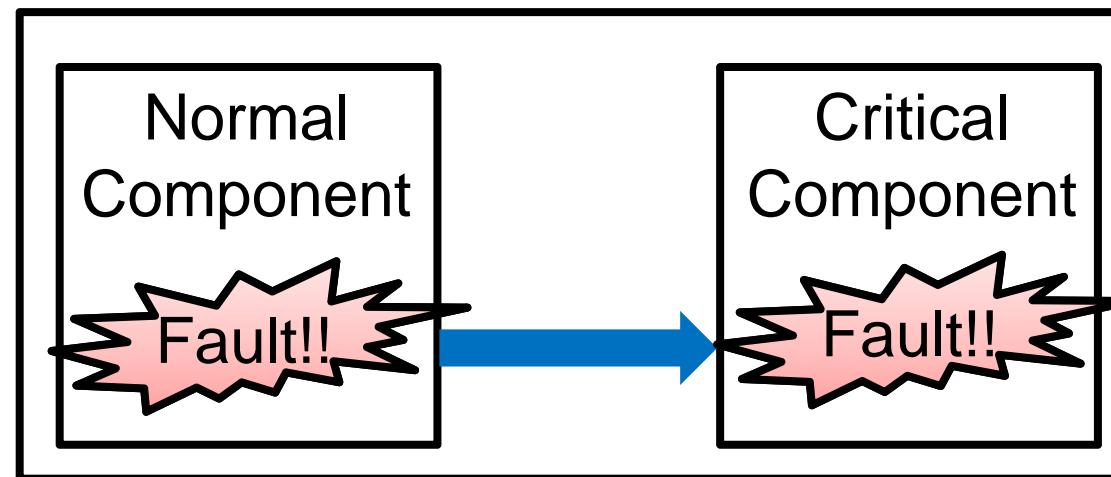
■ Efficiency (Right Axis) [TOPS/W]

Outline

- Background
- High Performance with Low Power Consumption
- **Functional Safety**
 - Requirements & Approaches
 - Partitioning
 - Control of ISP with runtime BIST
- Implementation Results
- Conclusion

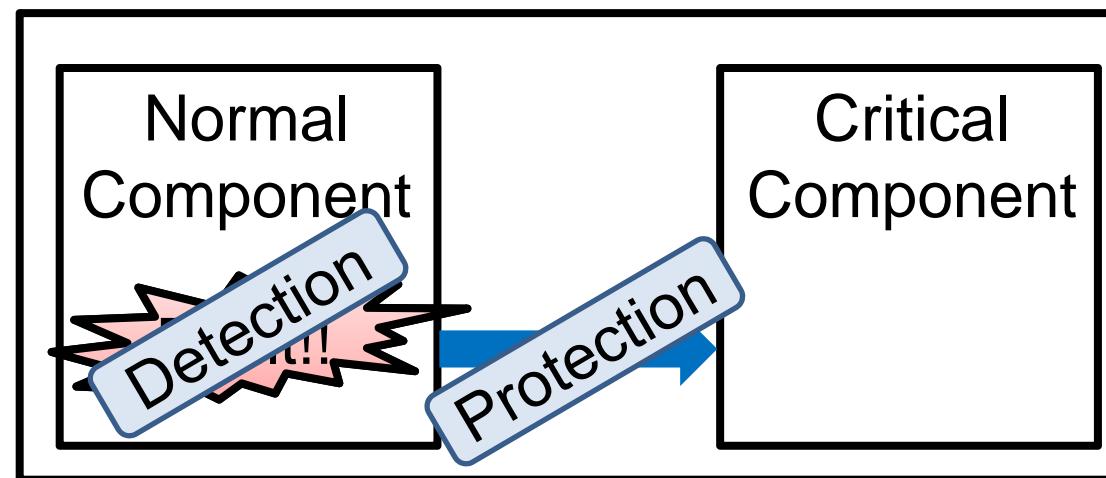
Requirement #2 : Functional Safety

- SoCs for ADAS require high safety to avoid serious accidents.
- When a component in SoC is faulty, the SoC has to ...
 - Detect the fault.
 - Continue to operate until reaching safe state.
 - Especially, the critical components like the central control system



Approaches for Functional Safety

- Goal of functional safety
 - Reduce the risk of accidents to acceptable levels
- Safety mechanisms (SMs) are needed to reduce the risk
 - Partitioning critical regions to prevent fault propagation
 - Diagnostic features to detect faults



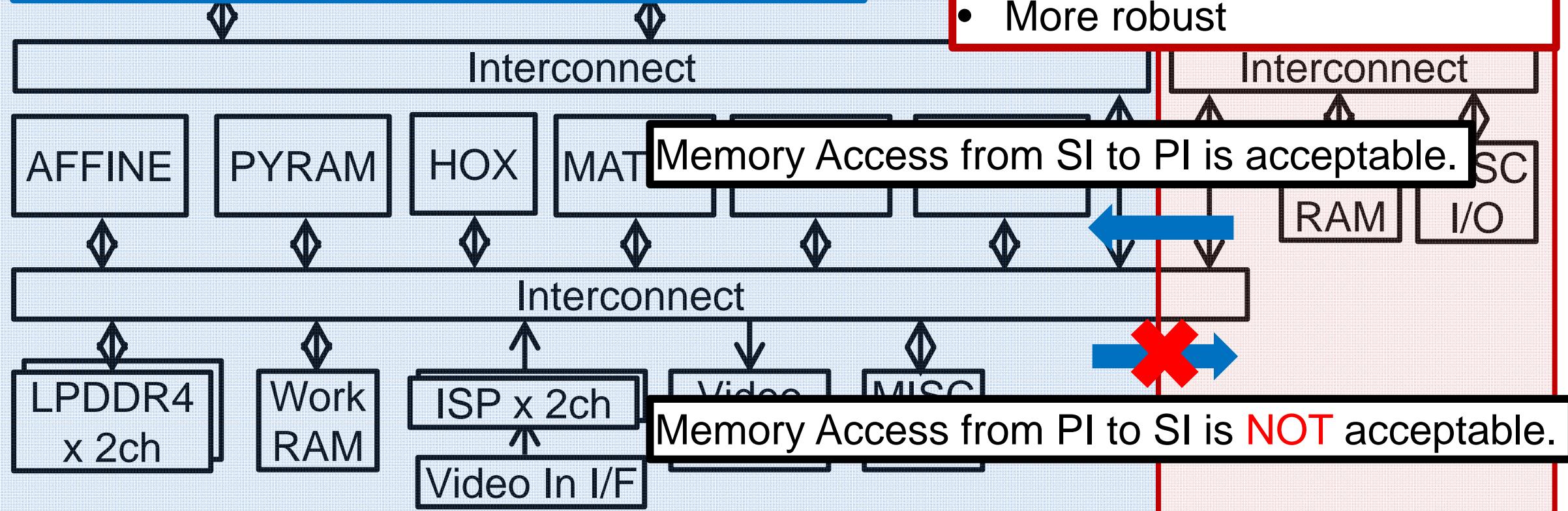
Critical Region Partitioning

Processing Island (PI)

- Executes image recognition applications
- Requires high performance and low power
- Complies with ASIL-B

Safe Island (SI)

- Executes control applications
- Requires high safety
- Complies with ASIL-D
- More robust



Safety Mechanisms for Fault Detection

Processing Island (PI)

- Executes image recognition applications
- Requires high performance and low power
- Complies with ASIL-B

Safe Island (SI)

- Executes control applications
- Requires high safety
- Complies with ASIL-D
- More robust

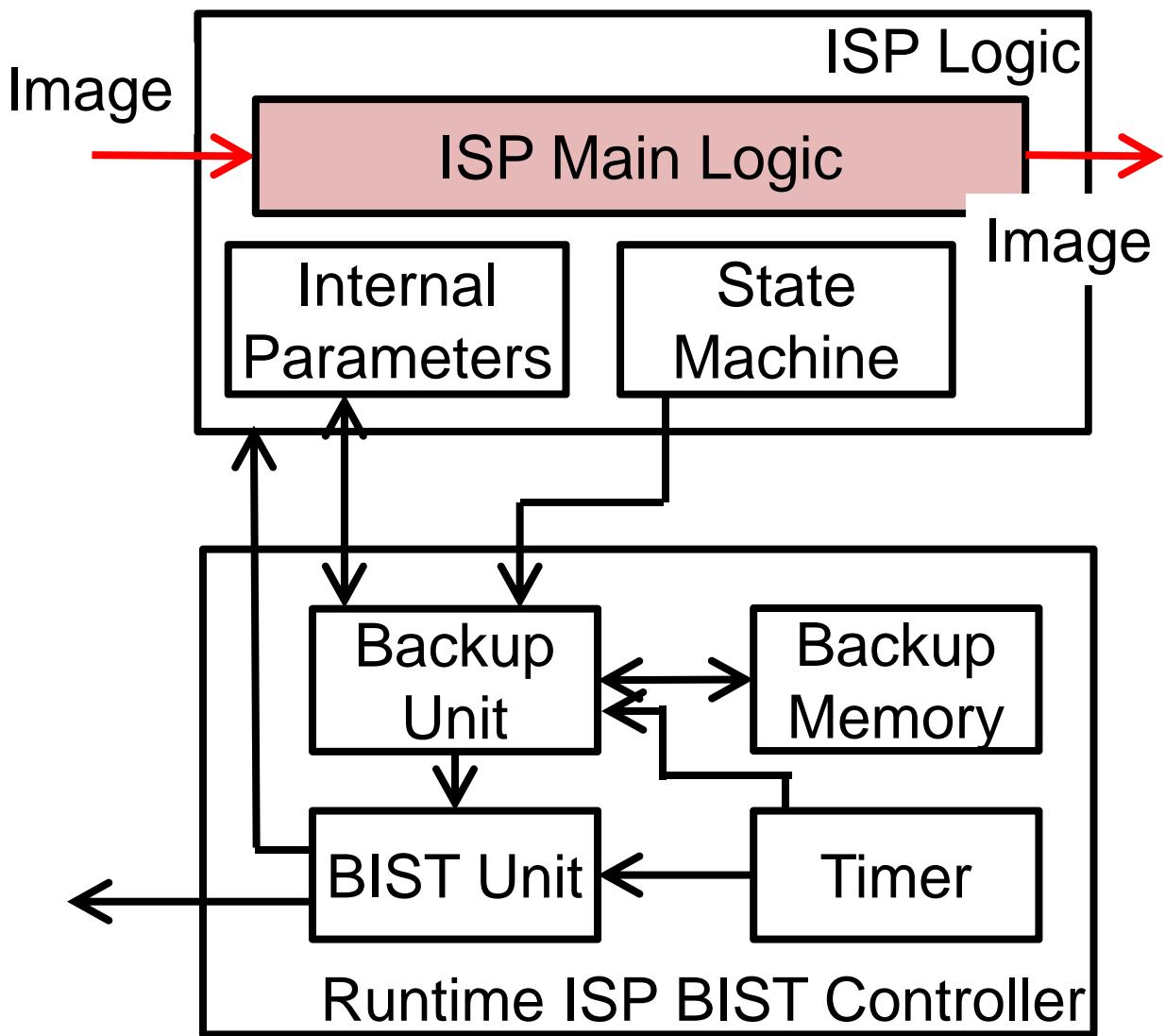
Target of Fault Detection	PI complies with ASIL-B	SI complies with ASIL-D
Random Logic	Runtime BIST*	Duplicated Logic
Memory Logic	Parity / ECC	ECC
Memory Access	MPU	Duplicated MPU
Clock, Voltage, etc	Monitor	Duplicated Monitor
Bus Access	ECC with bus payload	ECC with bus payload

*Built-In Self Test (BIST)

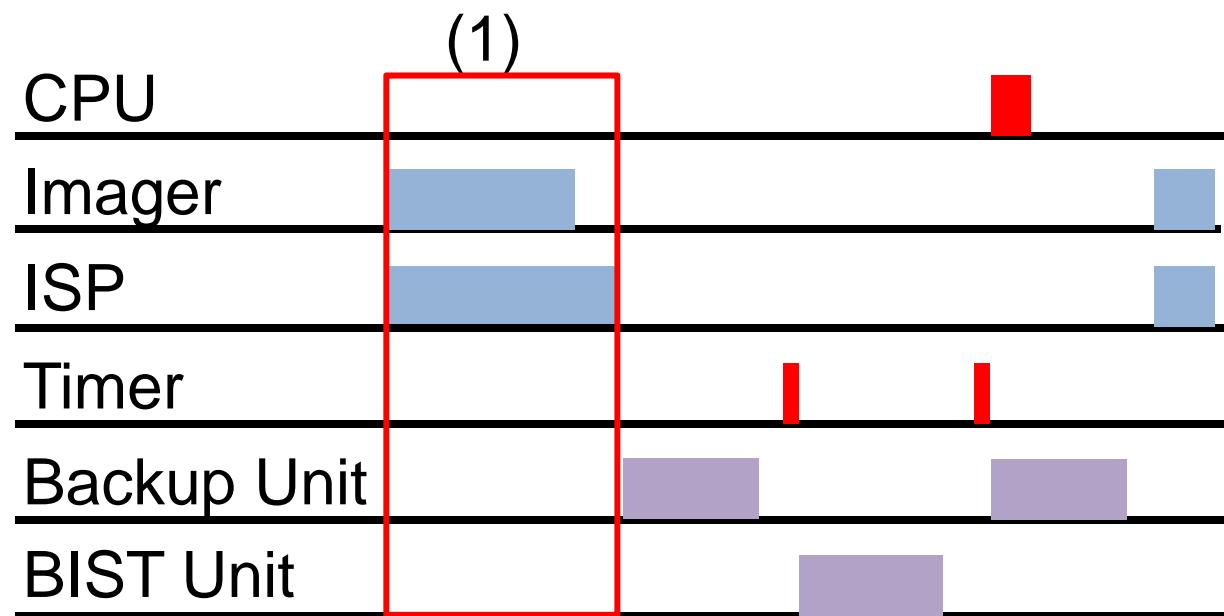
Control of ISP including Runtime BIST

- Processor controls HWAs including runtime BIST operation
- However, ISP is difficult to be controlled with processor
 - **External imager controls**, not processor, start timing
- Requirements
 - Only during V blanking period, BIST has to be running
 - Dedicated runtime ISP BIST controller
 - All parameters have to be re-configured after BIST
 - Backing up and restoring before and after BIST

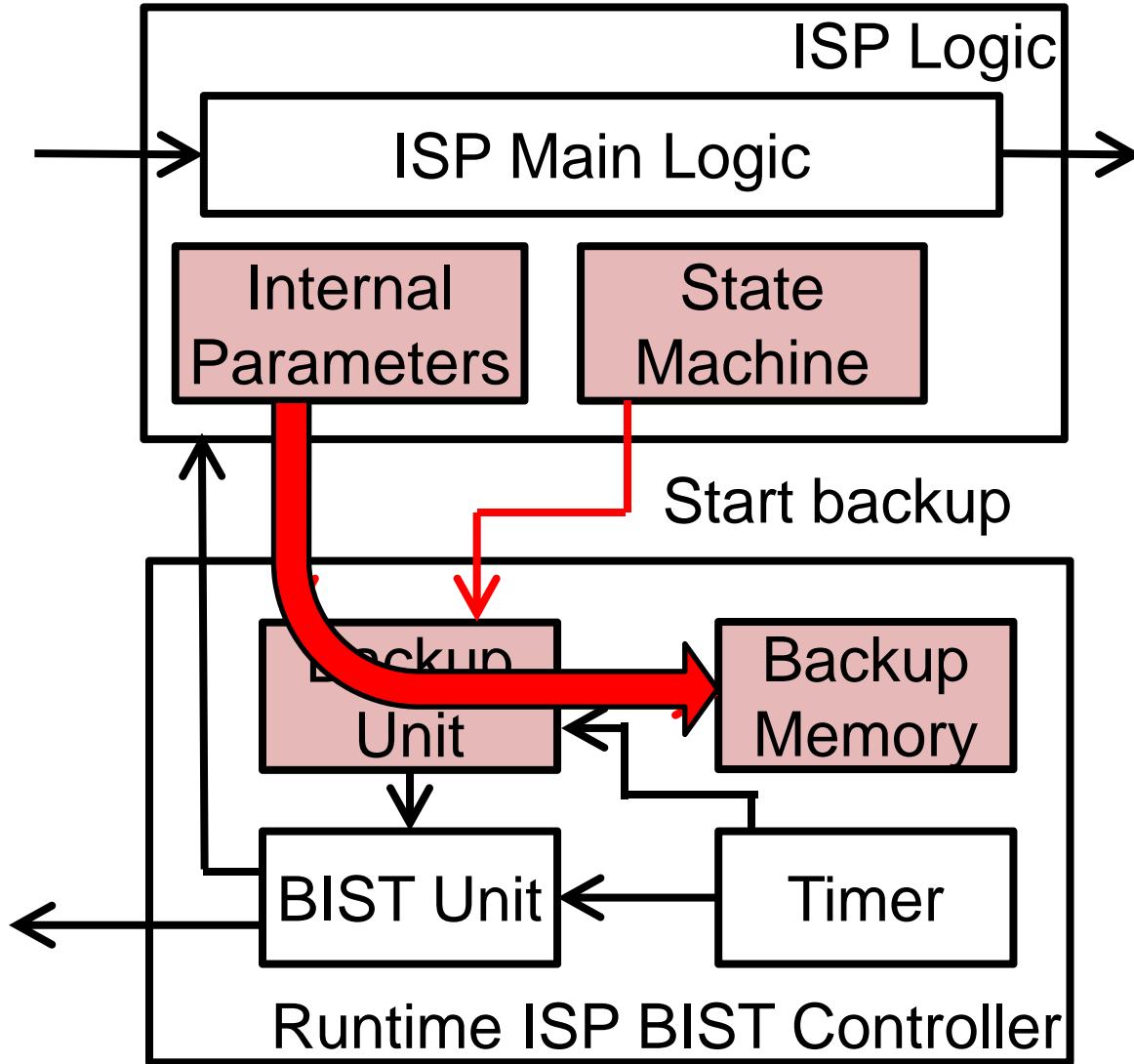
Control of ISP including Runtime BIST



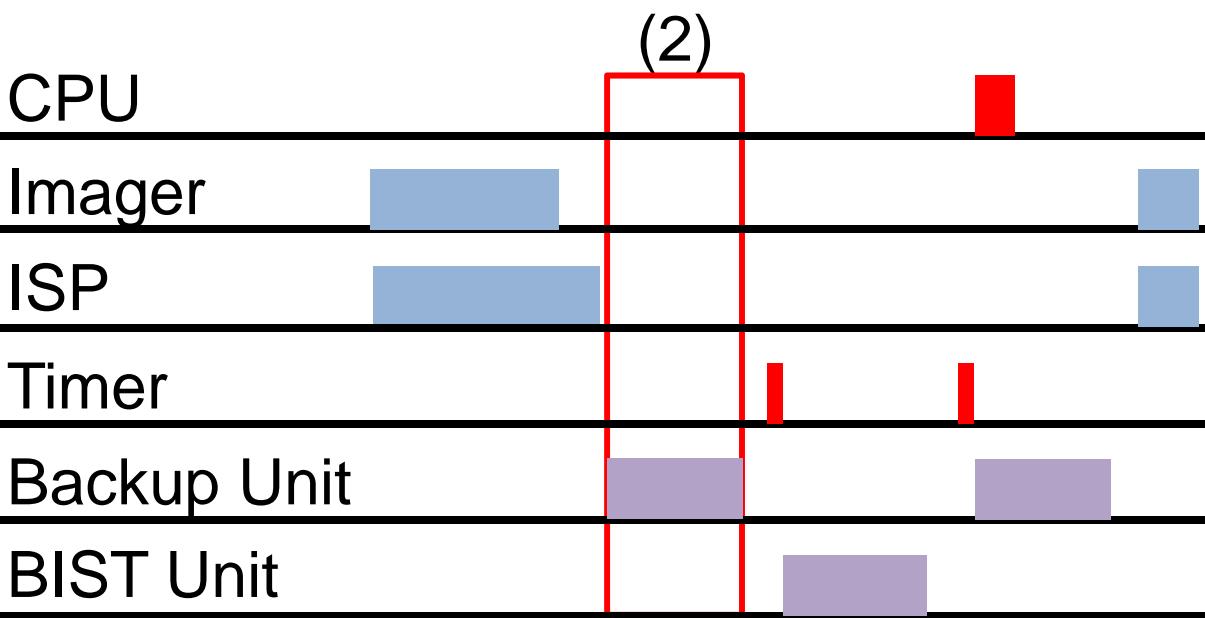
1. ISP receives new frame & starts its operation



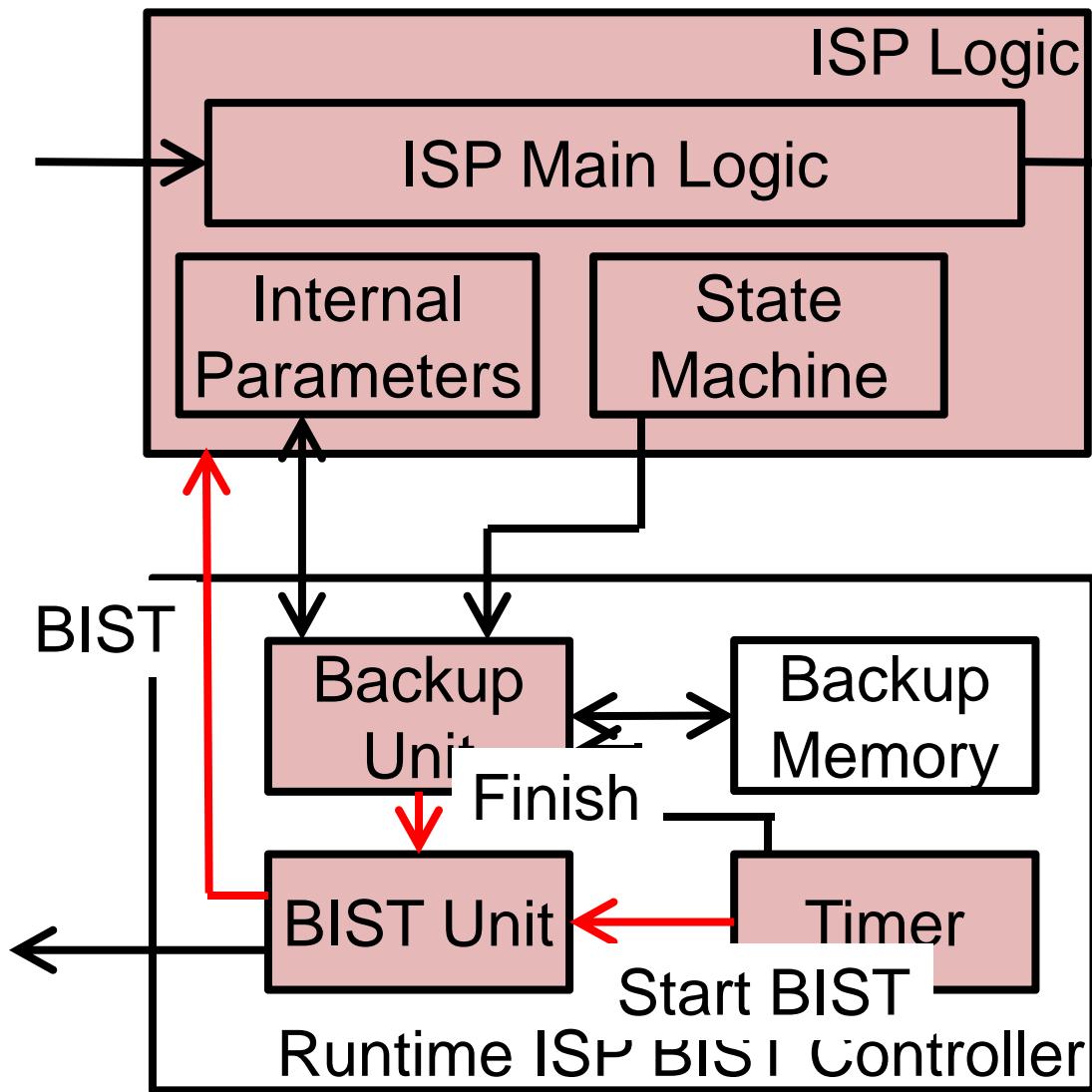
Control of ISP including Runtime BIST



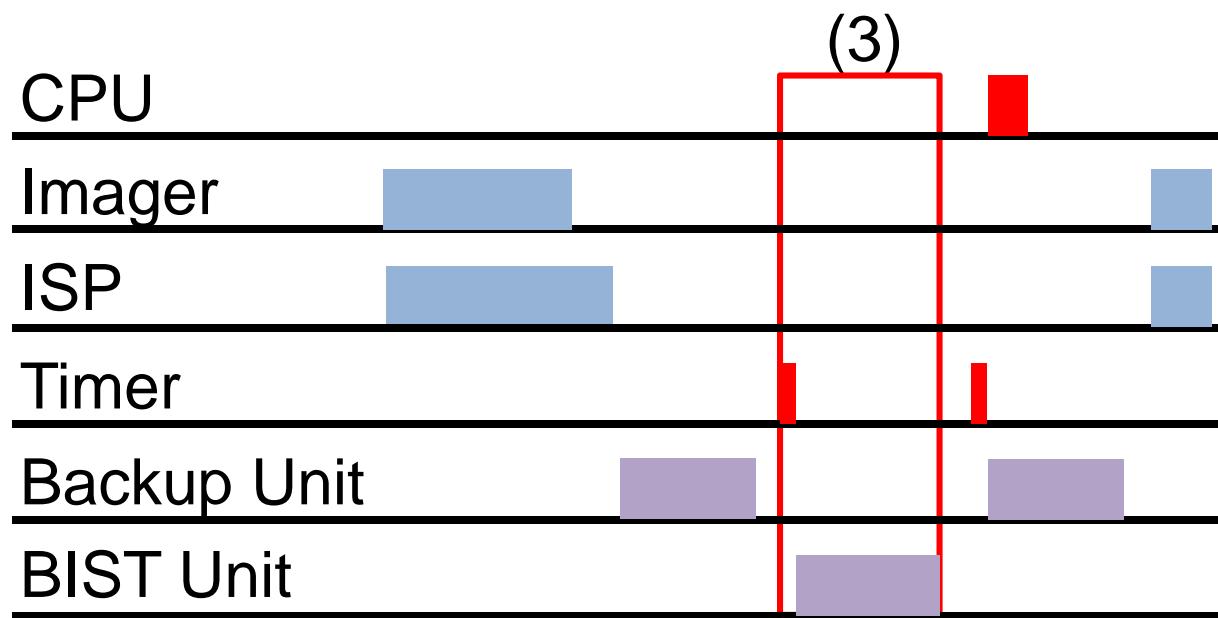
1. ISP receives new frame & starts its operation
2. Backup unit backs up all parameters



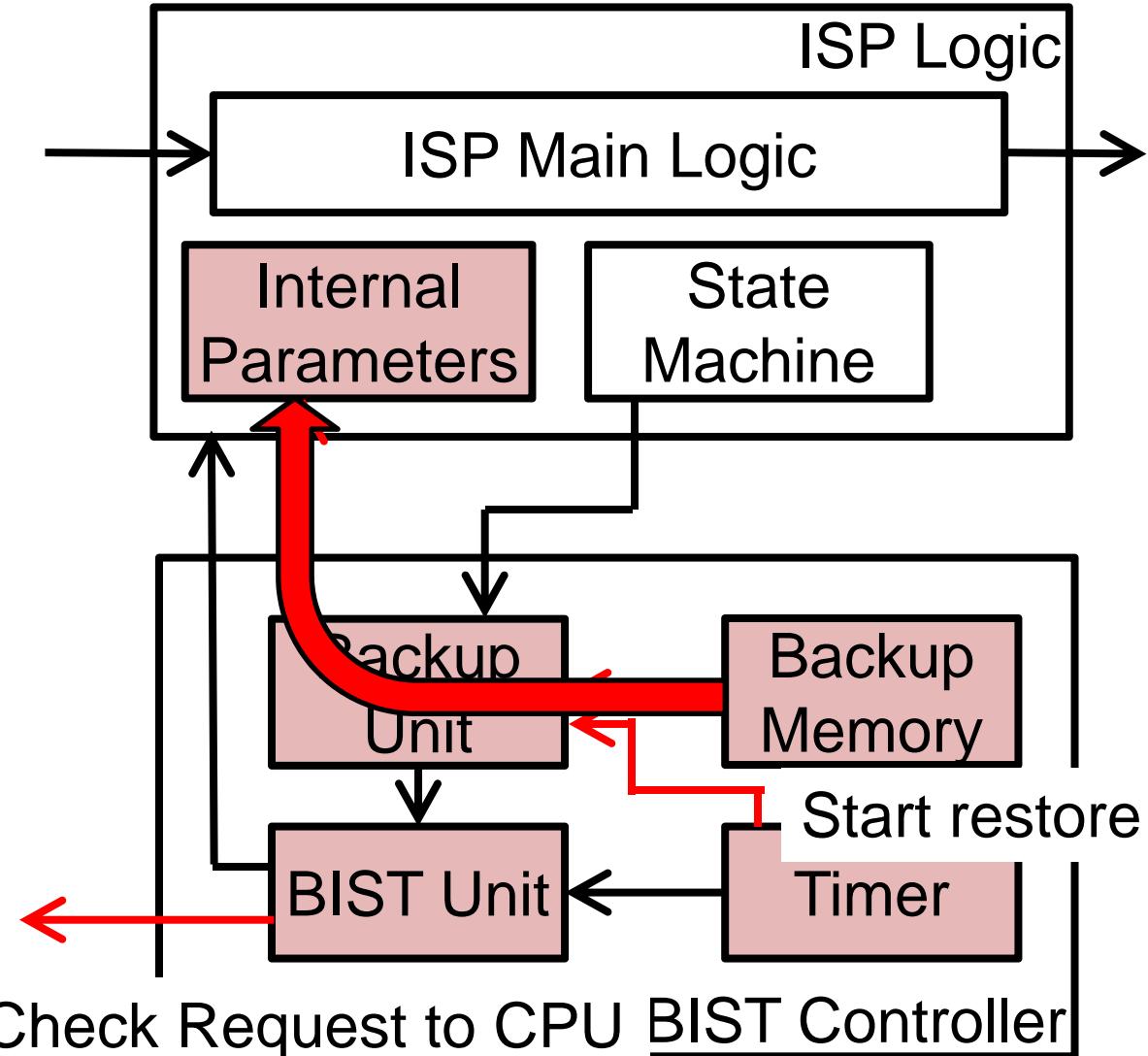
Control of ISP including Runtime BIST



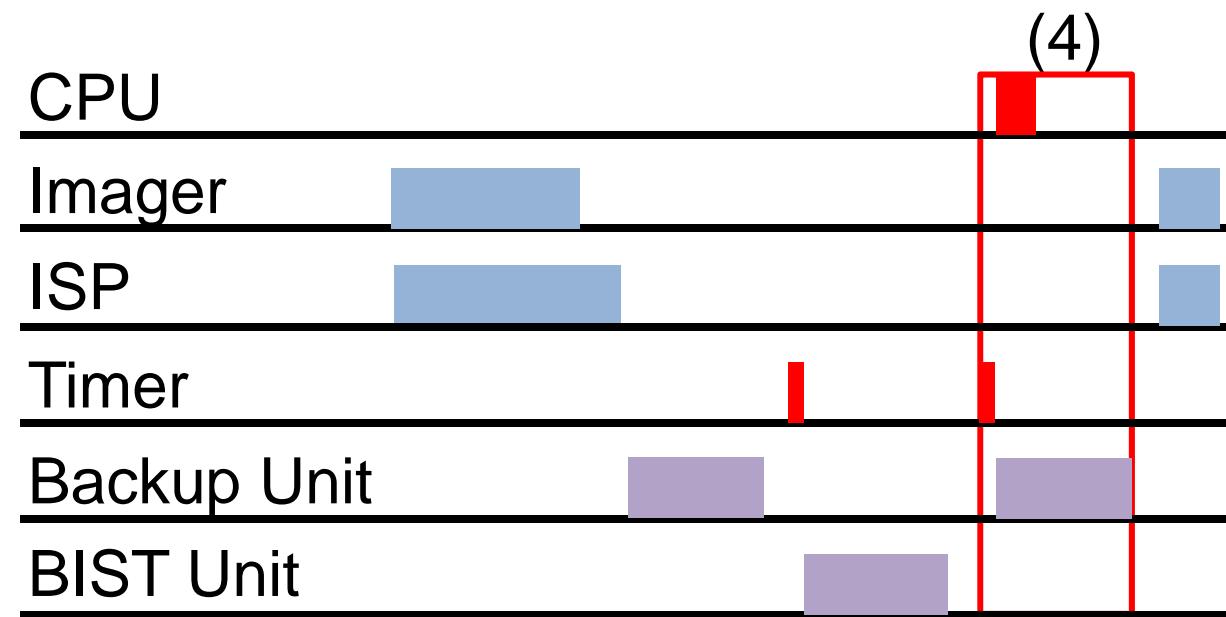
1. ISP receives new frame & starts its operation
2. Backup unit backs up all parameters
3. **BIST operation starts** if backup ends before Timer indicates



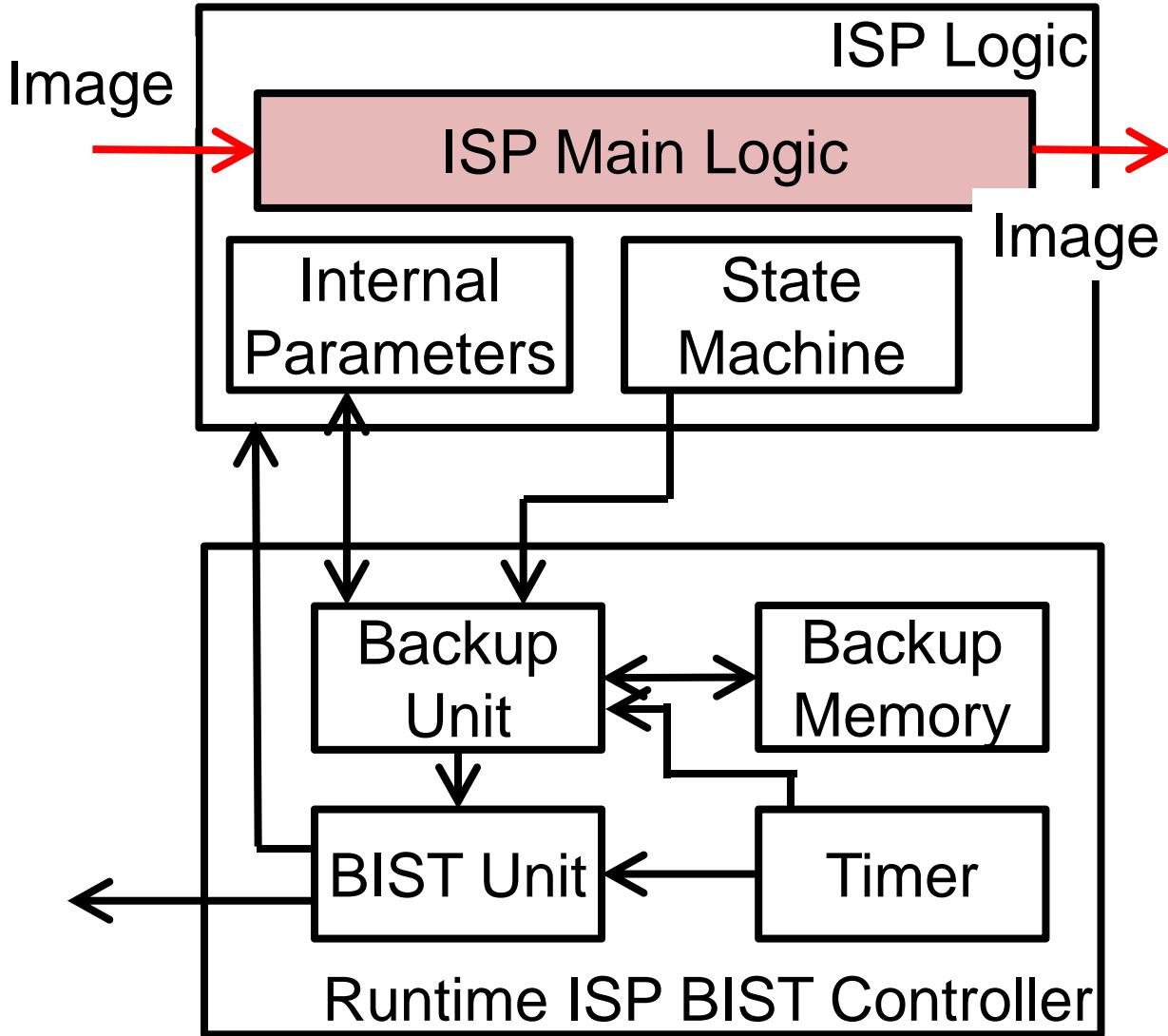
Control of ISP including Runtime BIST



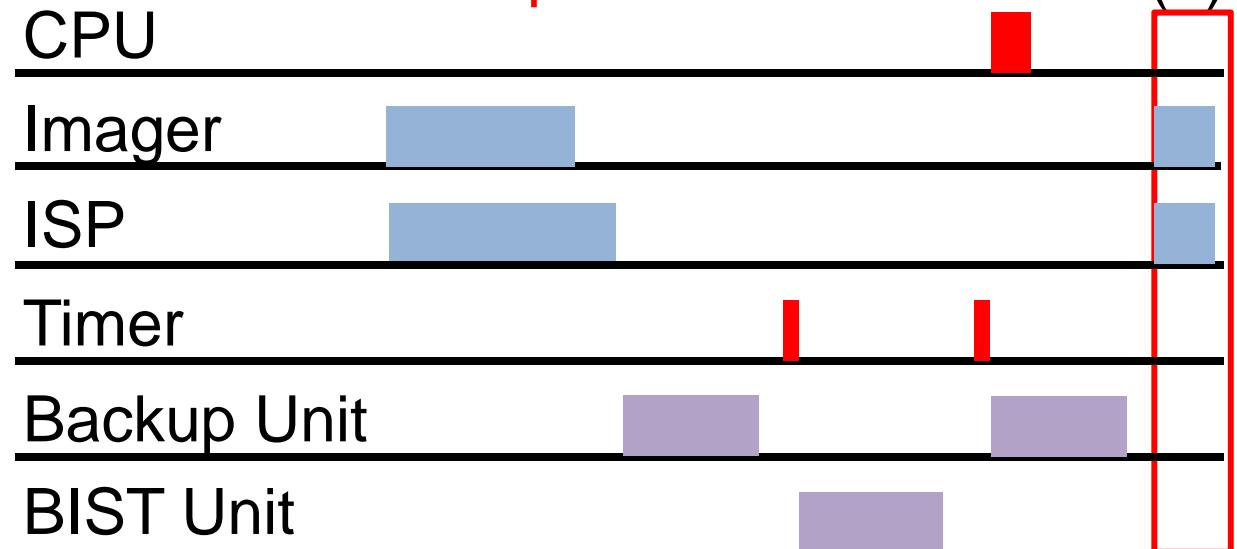
1. ISP receives new frame & starts its operation
2. Backup unit backs up all parameters
3. BIST operation starts if backup ends before Timer indicates
4. **CPU checks BIST result & Backup unit restores all parameters**



Control of ISP including Runtime BIST



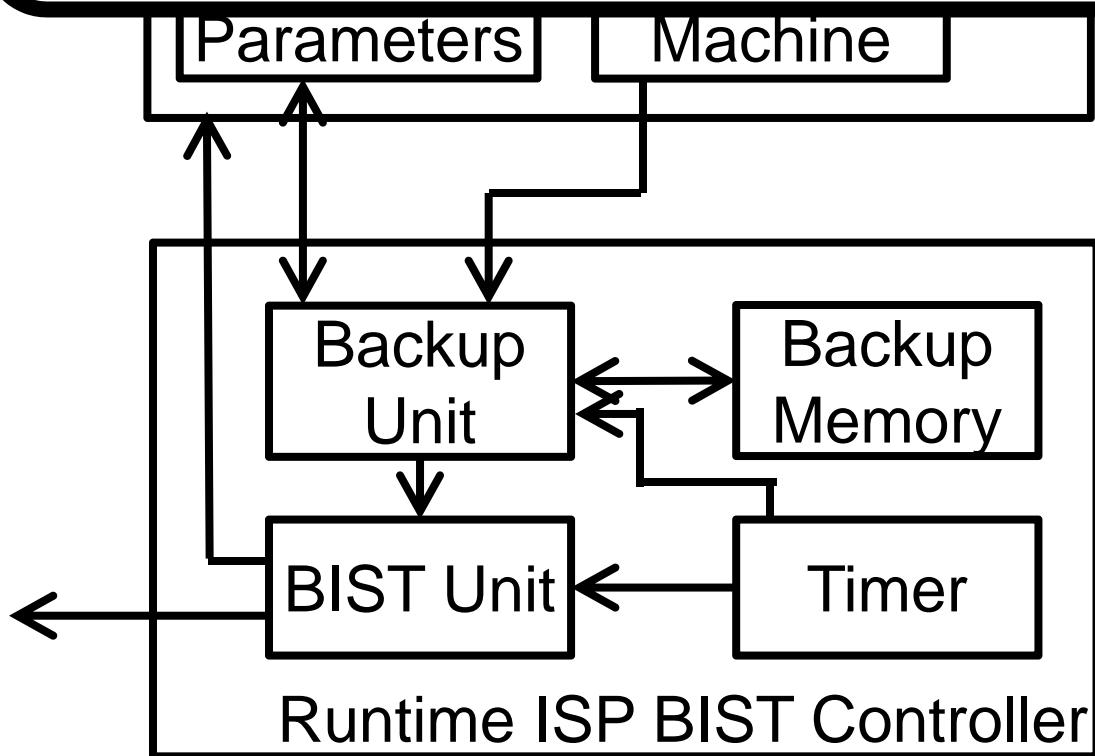
1. ISP receives new frame & starts its operation
2. Backup unit backs up all parameters
3. BIST operation starts if backup ends before Timer indicates
4. CPU checks BIST result & Backup unit restores all parameters
5. Continue ISP operation for next frame



Control of ISP including Runtime BIST

1. ISP receives new frame &

Runtime ISP BIST Controller can perform backup, BIST, and restore during V blanking period



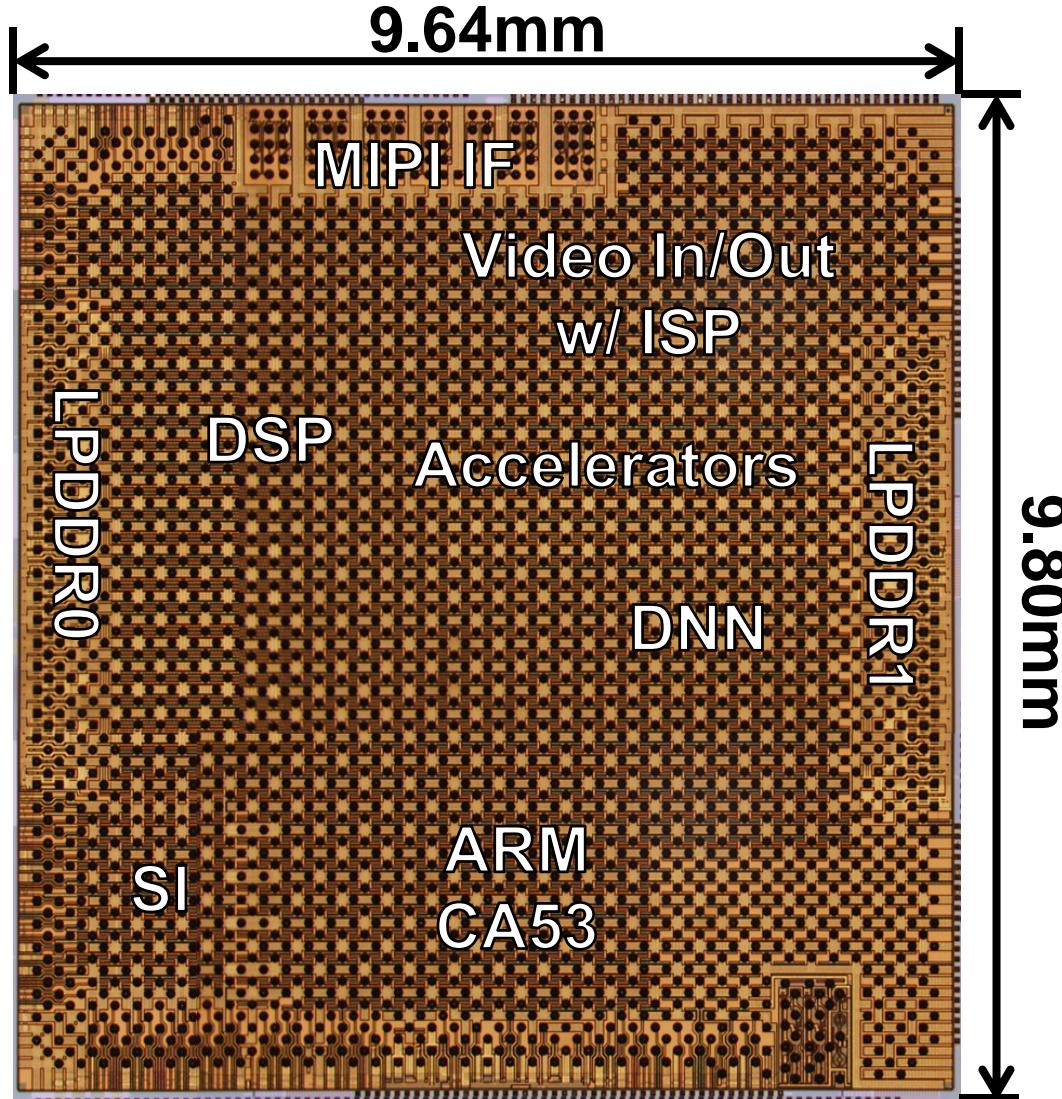
4. CPU checks BIST result & Backup unit restores all parameters
5. Continue ISP operation for next frame



Outline

- Background
- High Performance with Low Power Consumption
- Functional Safety
- **Implementation Results**
 - Chip Micrograph
 - Power Consumption
 - Demonstration
- Conclusion

Chip Micrograph



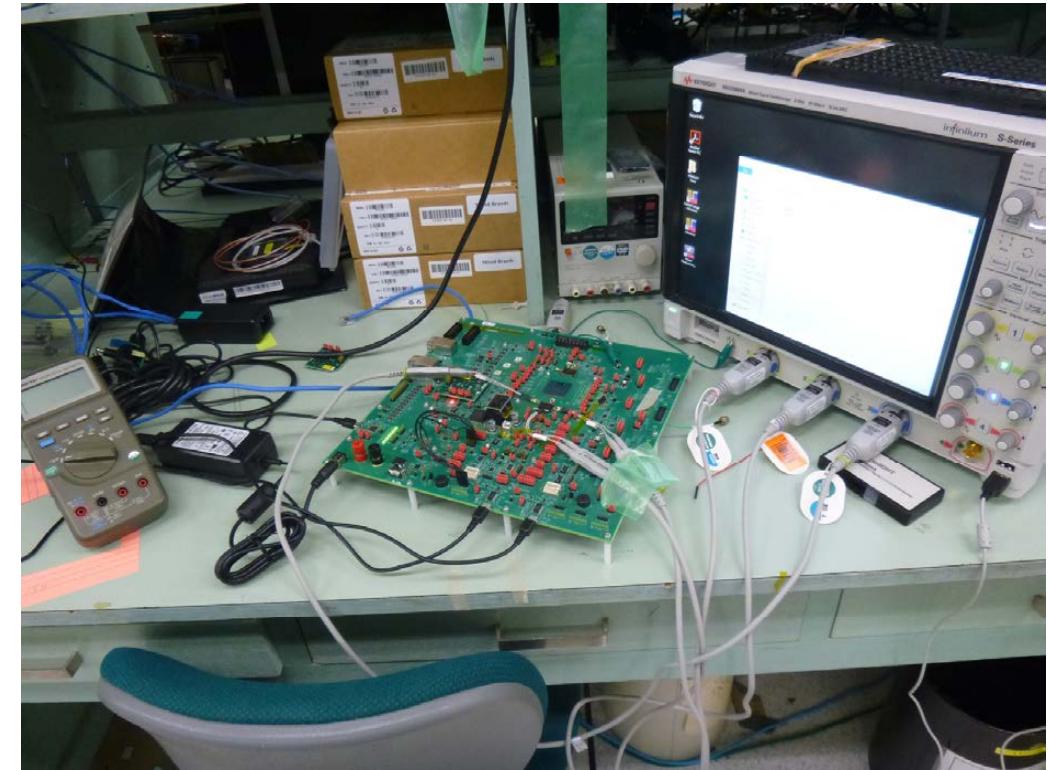
Process	16nm
Chip Size	94.52mm ²
On-Chip SRAM	142.9Mbit

Supply Voltage	
SI Core	0.8V
SI I/O	1.8~3.3V
PI Core	0.8V
PI I/O	1.8~3.3V

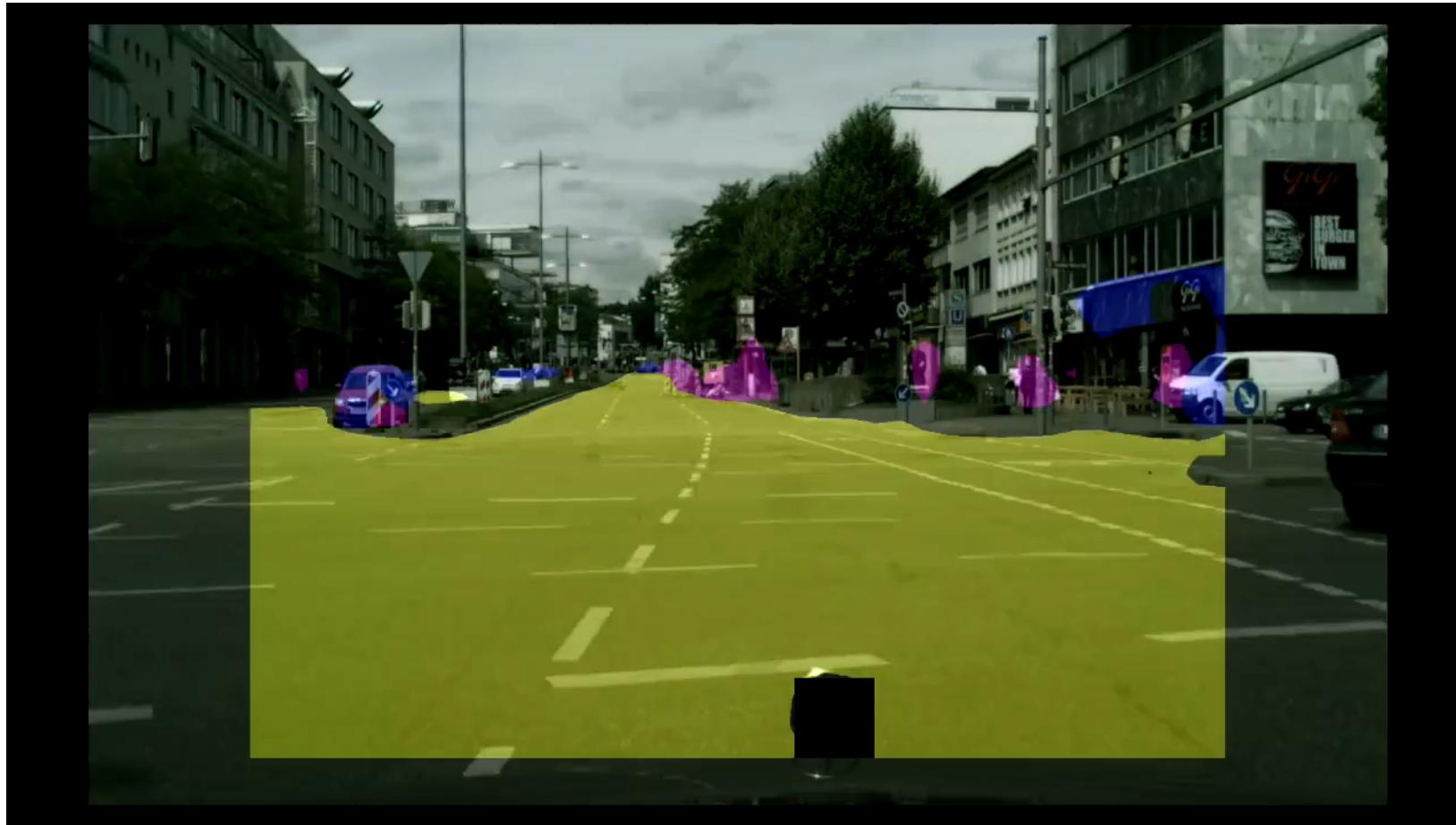
Power Consumption on Evaluation Board

- The power consumption of PI is 2.73W
- Executing 5 applications using one video stream
 - Pedestrian detection
 - Vehicle detection
 - Traffic signal recognition
 - Lane detection
 - Head light detection
- Running Resources
 - 2 cores of CA53
 - 4 DSPs
 - 6 types of accelerators: DNN, HOX, AFFINE, AKAZE, MATCH, PYRAM

Power Evaluation Environment



Demonstration of DNN Application



This Dataset is “Cityscapes Dataset”

Reference: M. Cordts, et.al, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Outline

- Background
- High Performance with Low Power Consumption
- Functional Safety
- Implementation Results
- Conclusion

Conclusion

- **We implemented the SoC for ADAS applications** to realize high performance with low power consumption and high safety
- Heterogeneous architecture achieves over 20 TOPS and 2TOPS/W
 - 8 processors and 4 DSPs
 - 8 types of hardware accelerators such as DNN and ISP
 - Power consumption of PI is 2.73 W by running use case on the evaluation board
- Safety mechanisms are introduced for high safety
 - e.g. partitioning (PI and SI) and runtime BIST

Thank you for your attention!!

An 879GOPs 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration

Ziyun Li, Yu Chen, Luyao Gong, Lu Liu, Dennis Sylvester, David Blaauw, Hun-Seok Kim
University of Michigan, Ann Arbor, MI

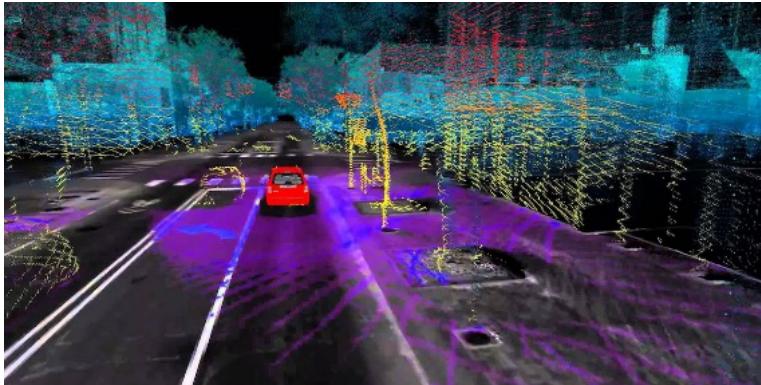


Application of SLAM

- SLAM (simultaneous localization and mapping)
 - Estimation of 3D position & 3D movement
 - Constructs a 3D map of an unknown surrounding
 - Fundamental kernel for autonomous systems



“Where I am?”



Self-driving cars



Micro Aerial Vechicle



VR / AR

Constraints of Mobile SLAM

- SLAM (simultaneous localization and mapping)
 - Estimation of 3D position & 3D movement
 - Constructs a 3D map of an unknown surrounding
 - Fundamental kernel for autonomous systems

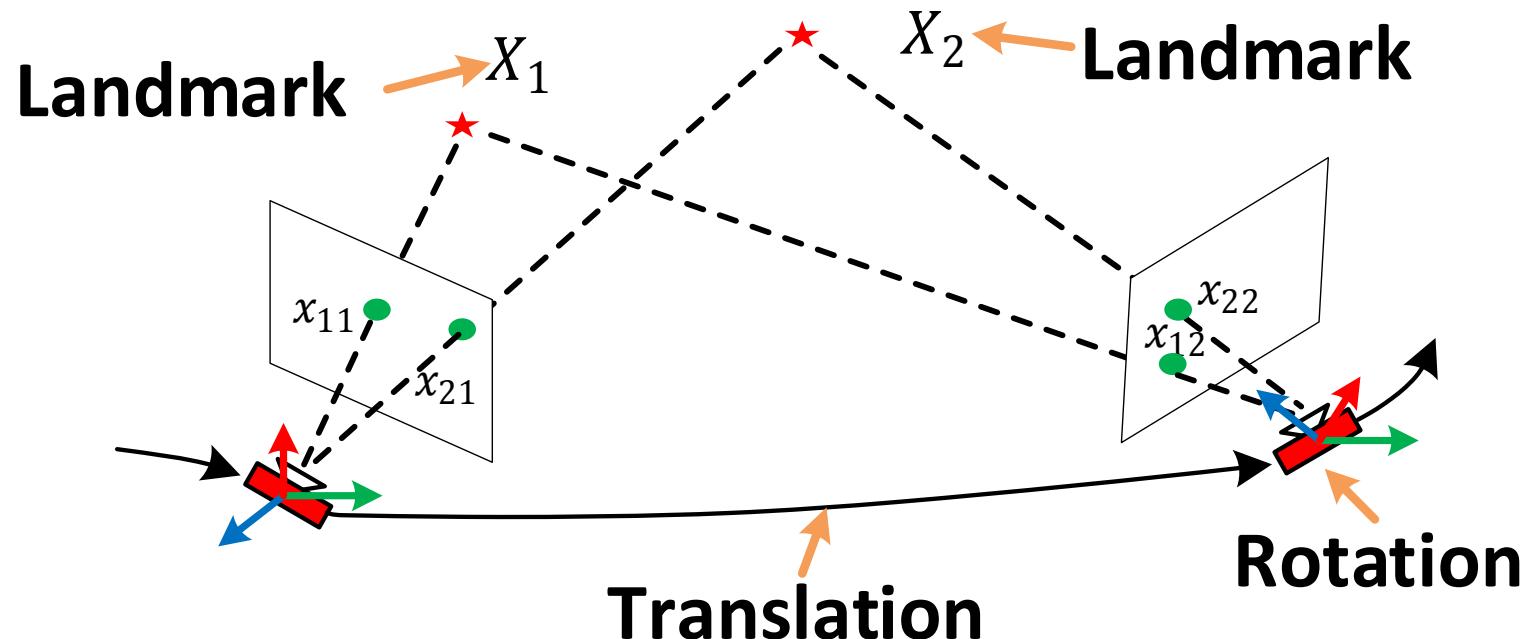
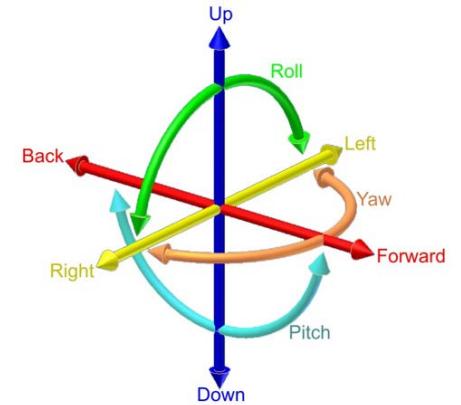


Micro Aerial Vechicle

- Constraints on mobile systems
 - Robust (wide range: $\pm 1\text{km}$, rapid movement: 10MPH)
 - High performance ($\sim 30\text{ms}$ response latency, or faster)
 - Accurate (low drift, $< 1\%$ translation & rotation error)
 - SWaP requirement:
 - Low Power ($\sim 100\text{mW}$); Size: small ($\sim 50\text{cm}^3$); Weight: light ($\sim 100\text{g}$)

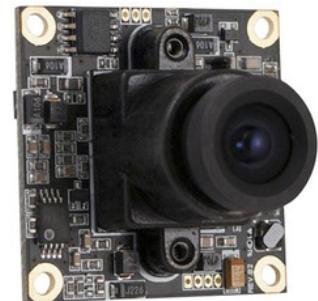
Introduction of SLAM

- Same landmarks can be perceived from different viewpoints
 - Constructs a 3D map of these landmarks
 - 3D position
 - Use constructed 3D map to estimate egomotion
 - $[R \mid T] \rightarrow 6 \text{ DoF pose} = 3\text{D rotation} + 3\text{D translation}$



Sensing modalities for SLAM

- Inertial Sensor (IMU)
 - Accurate IMU has >100mW power consumption
 - IMU has drift → requires filtering/sensor fusion
 - Compensate drift with camera-based approaches [VLSI'18]
- LIDAR
 - Depth point cloud
 - → Massive processing is still required
 - Large size (>500 cm³) & expensive
 - High power consumption (>3W)
- Cameras
 - Low cost & low power consumption (~100mW)
 - Readily available in many mobile systems



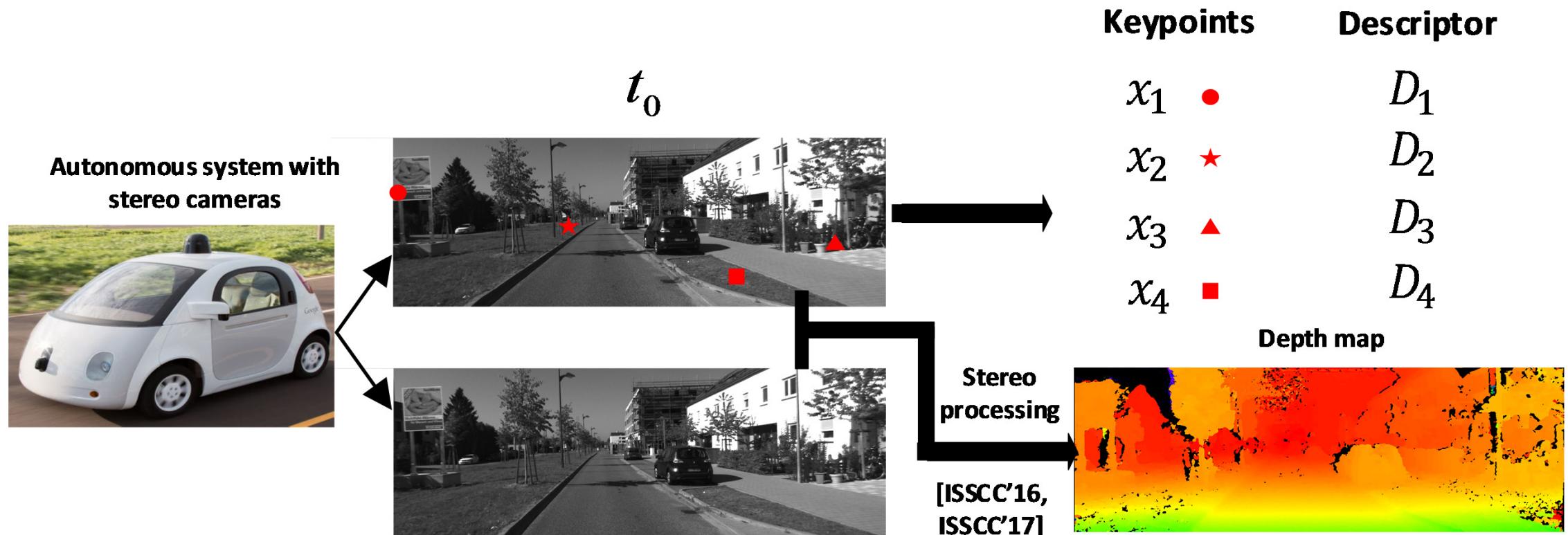
Sensing modalities for SLAM

- Inertial Sensor (IMU)
 - Accurate IMU has >100mW power consumption
 - IMU has drift → requires filtering/sensor fusion
 - Compensate drift with camera-based approaches [VLSI'18]
- LIDAR
 - Depth point cloud
 - → Massive processing is still required
 - Large size (>500 cm³) & expensive
 - High power consumption (>3W)
- Cameras
 - Low cost & low power consumption (~100mW)
 - Readily available in many mobile systems



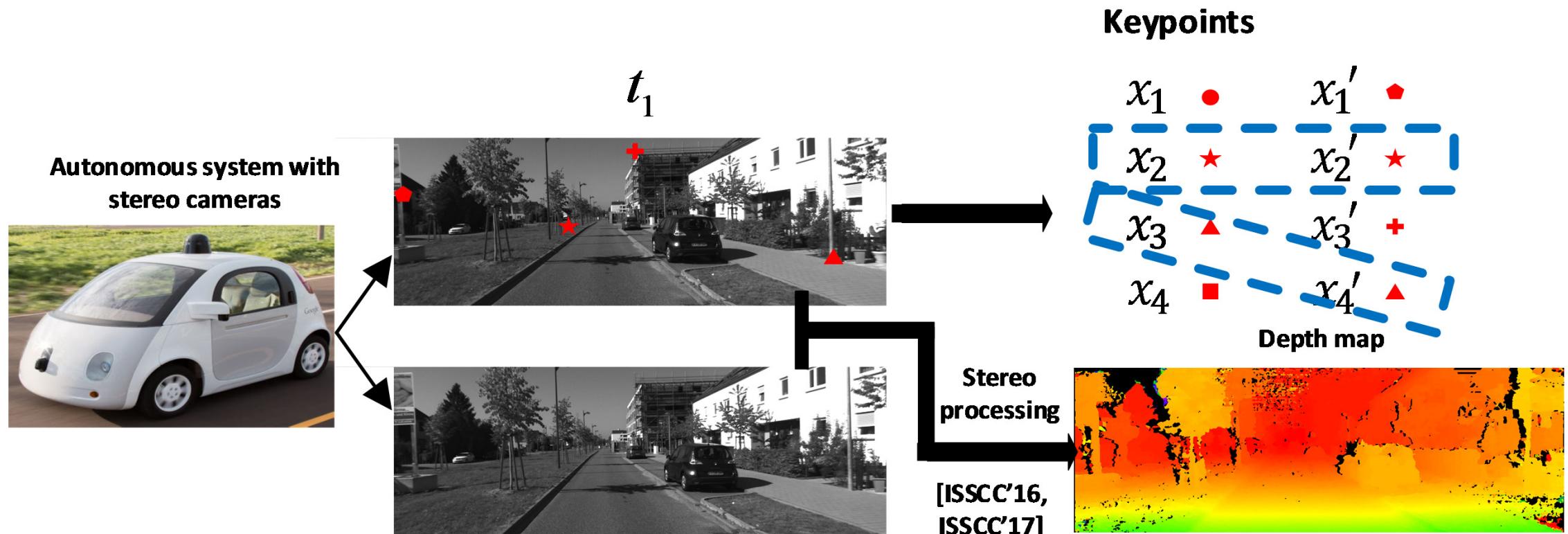
SLAM Processing: Keypoint Matching

- Extract keypoints from current frame @ t0
 - DoG, Harris, etc. → 3 level DoG keypoints are used in this design



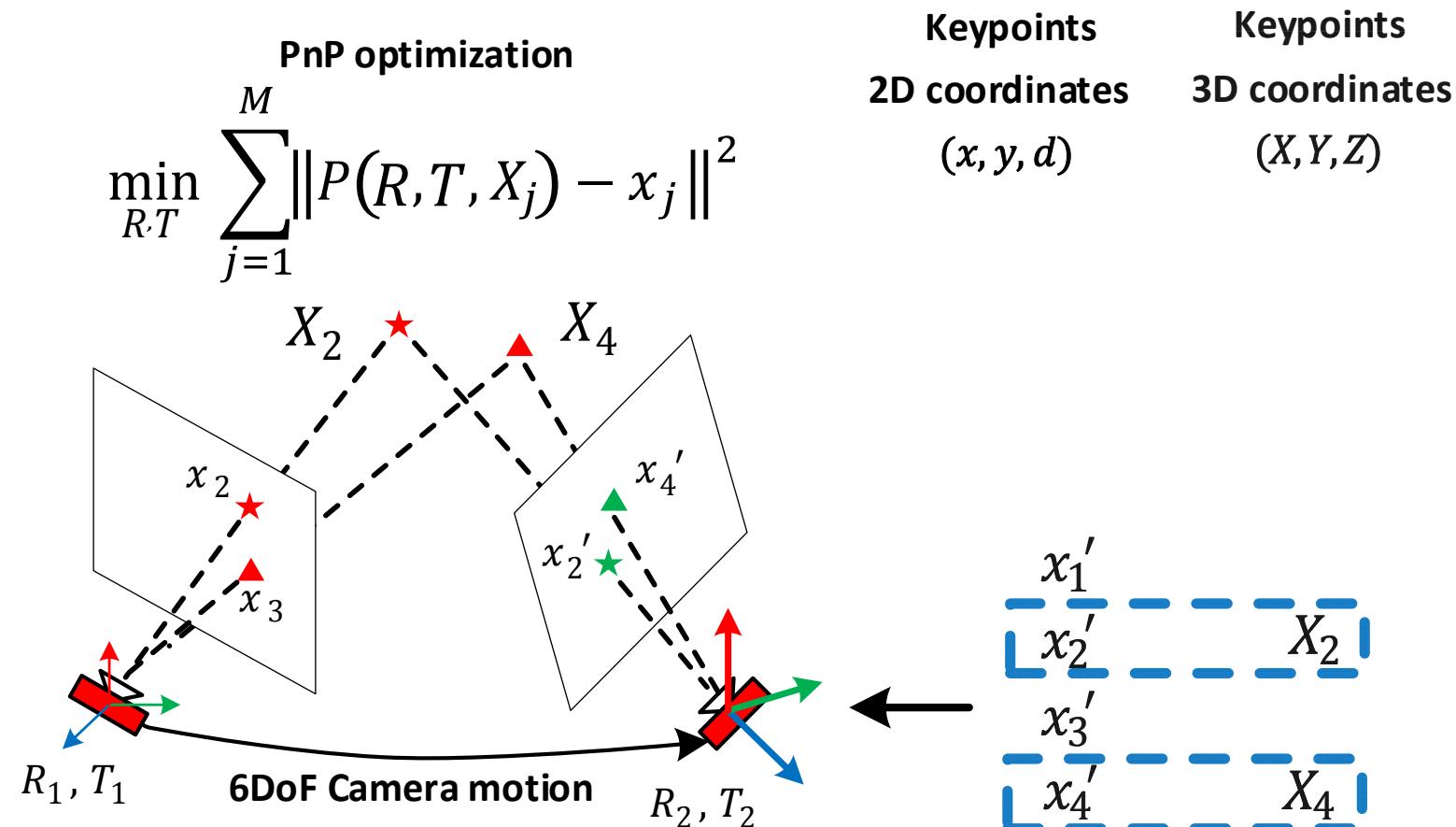
SLAM Processing: Keypoint Matching

- Extract keypoints from current frame @ t1
 - DoG, Harris, etc. → 3 level DoG keypoints are used in this design



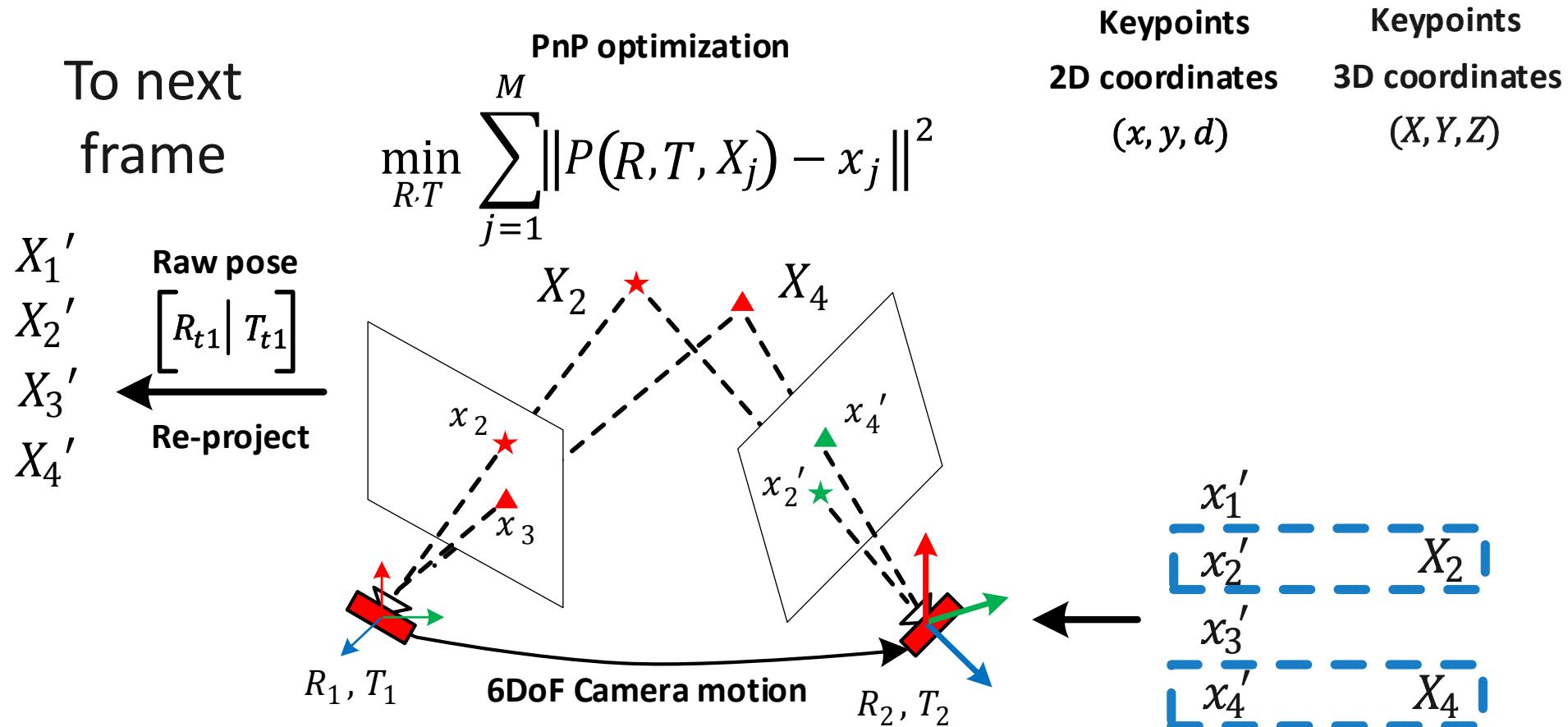
SLAM Processing: Perspective-n-Point(PnP)

- 6 DoF pose of current frame @ t1 is solved through optimization
 - With known 2D + 3D coordinate pairs



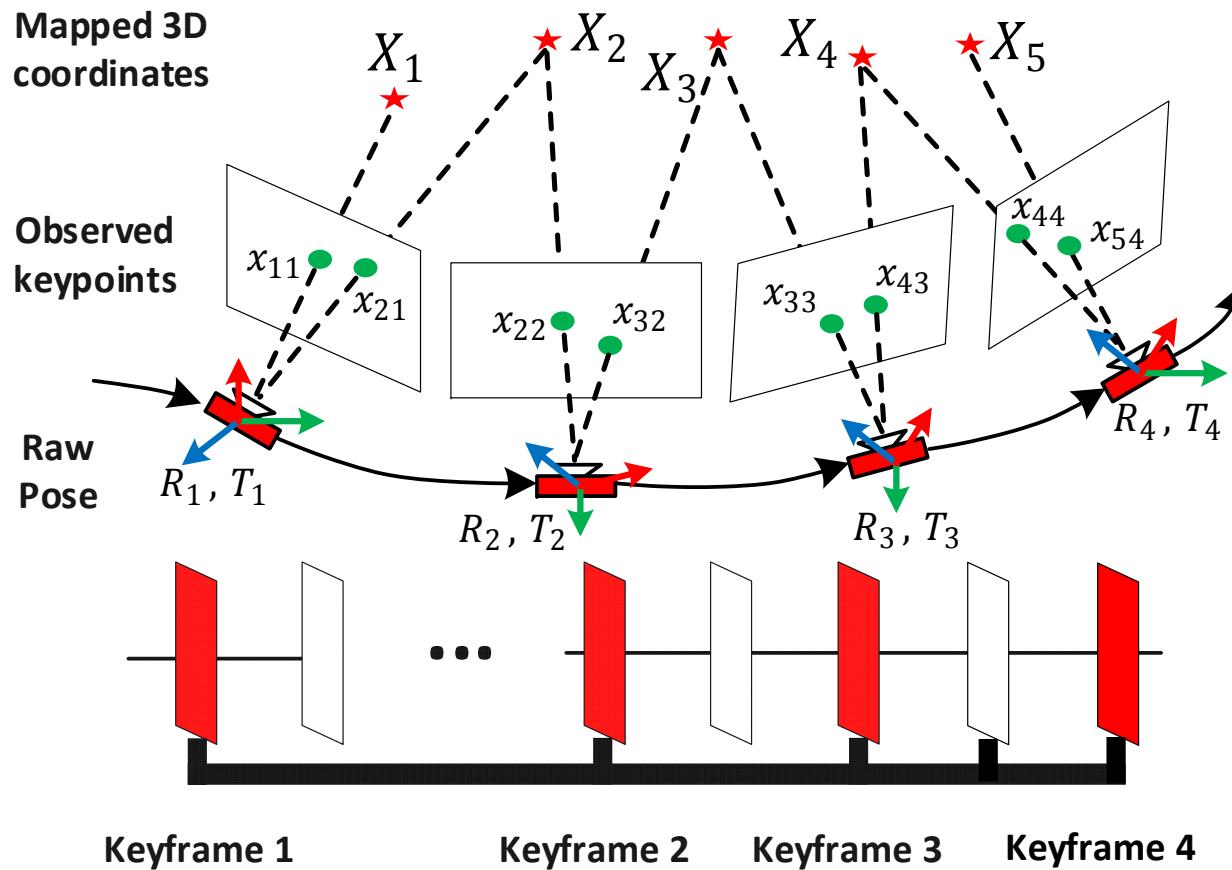
SLAM Processing: PnP

- 6 DoF pose of current frame @ t1 is solved through optimization
 - Re-project all keypoints from current frame t1 for processing next frame



SLAM Processing: Bundle Adjustment

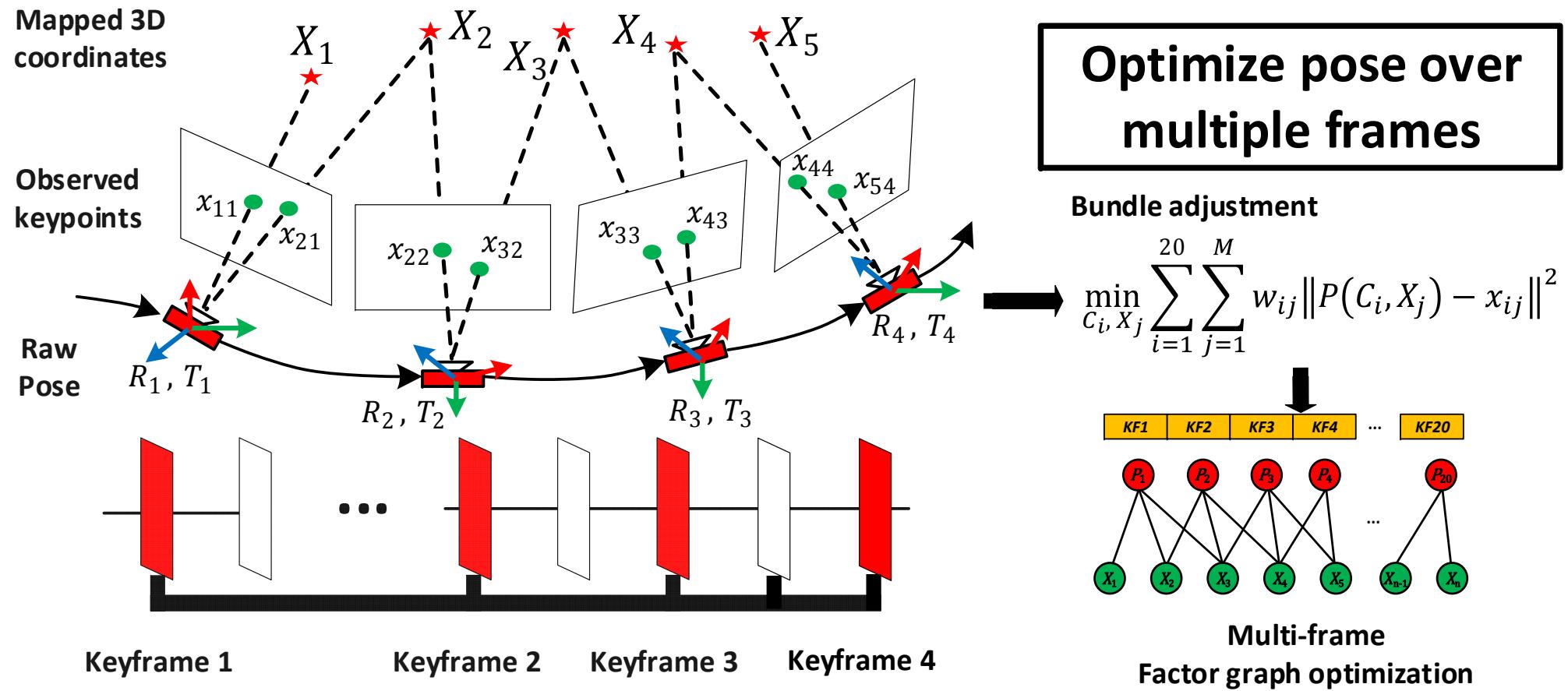
- Raw pose and matching pairs are tracked for multiple frames
 - Same 3D point maybe observed in multiple frames



Current frame is identified as keyframe if it tracks more than 50% **new points** than previous keyframe

SLAM Processing: Bundle Adjustment

- Raw poses and observation forms a factor graph
 - Jointly optimize pose of 20 keyframes through Levenberg-Marquart algorithm



Hardware Challenges

- Hardware cost of running SLAM 80fps VGA
 - >60 fps, ~1 Gb/s incoming image streams
 - >250 GOPs computation
 - Growing map size (>1000 points/frame)
- Heterogenous kernels and complex control flow
 - Vision front end: Keypoint detection, feature extraction, feature matching ...
 - Frame by frame tracking: Projection, matrix inversion, PnP, outlier rejection ...
 - Local bundle adjustment: Projection, Rodrigue, Large matrix solver ...
- High precision processing
 - Large dynamic range in computing jacobian, LM iteration ...
- Large problem dimension



Hardware Challenges

- Hardware cost of running SLAM 80fps VGA
 - >60 fps, ~1 Gb/s incoming image streams
 - >250 GOPs computation
 - Growing map size (>1000 points/frame)



- Low efficiency on CPU/GPU platforms

[L. Nardi, 2015]

System	GPU (TITAN X)	CPU (GTX870M)	GPU (TK1)	ODROID	Arndale
Performance @ VGA	135 fps	96 fps	22 fps	5.5 fps	4.2 fps
Power	~400W	~100W	~20W	~10W	~5W
Energy per frame	~2.9J	~1J	~0.9J	~1.8J	~1.1J
>External DRAM	Yes	yes	yes	yes	yes

Hardware Challenges

- Hardware cost of running SLAM 80fps VGA
 - >60 fps, ~1 Gb/s incoming image streams
 - >250 GOPs computation
 - Growing map size (>1000 points/frame)



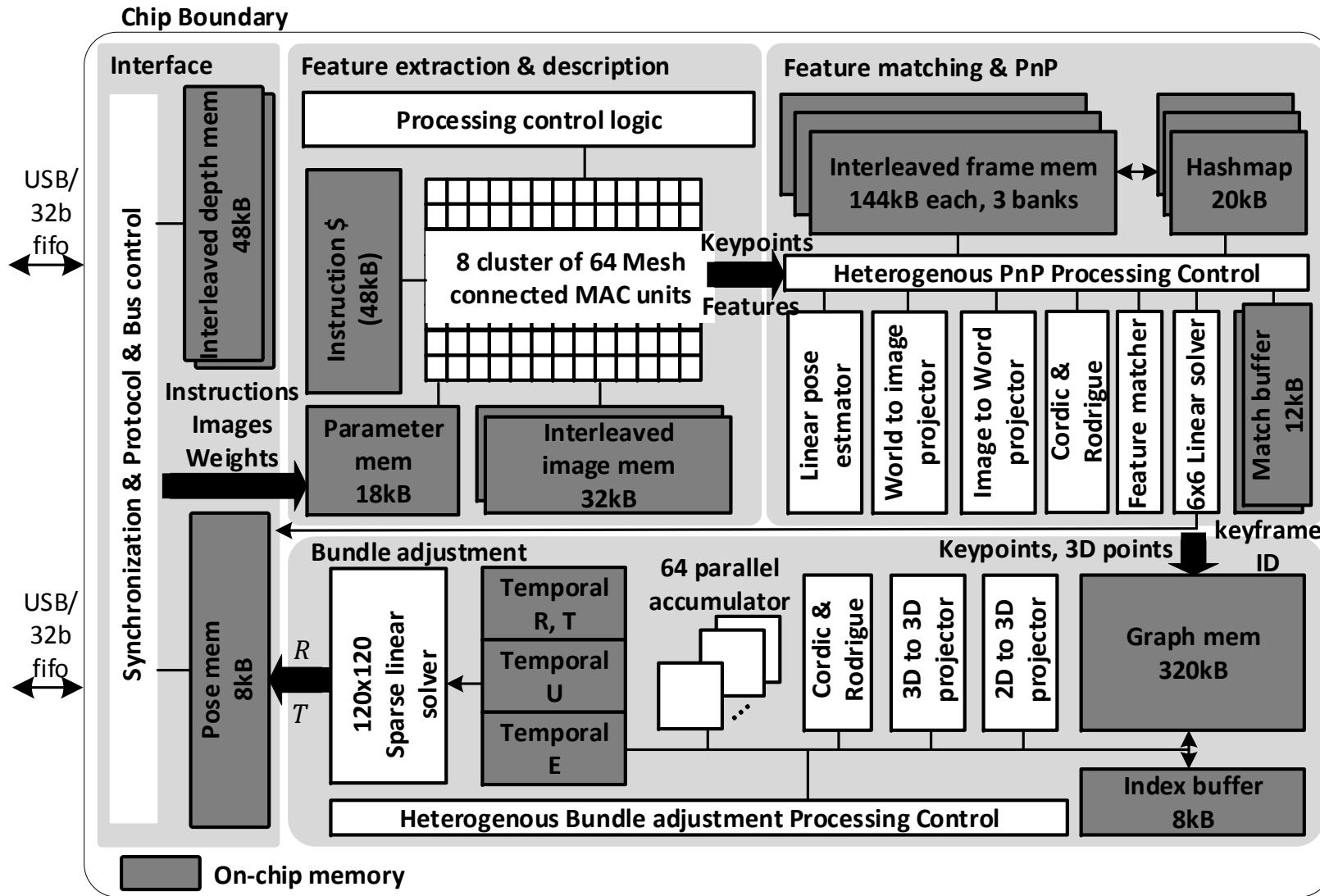
- Low efficiency on CPU/GPU platforms

[L. Nardi, 2015]

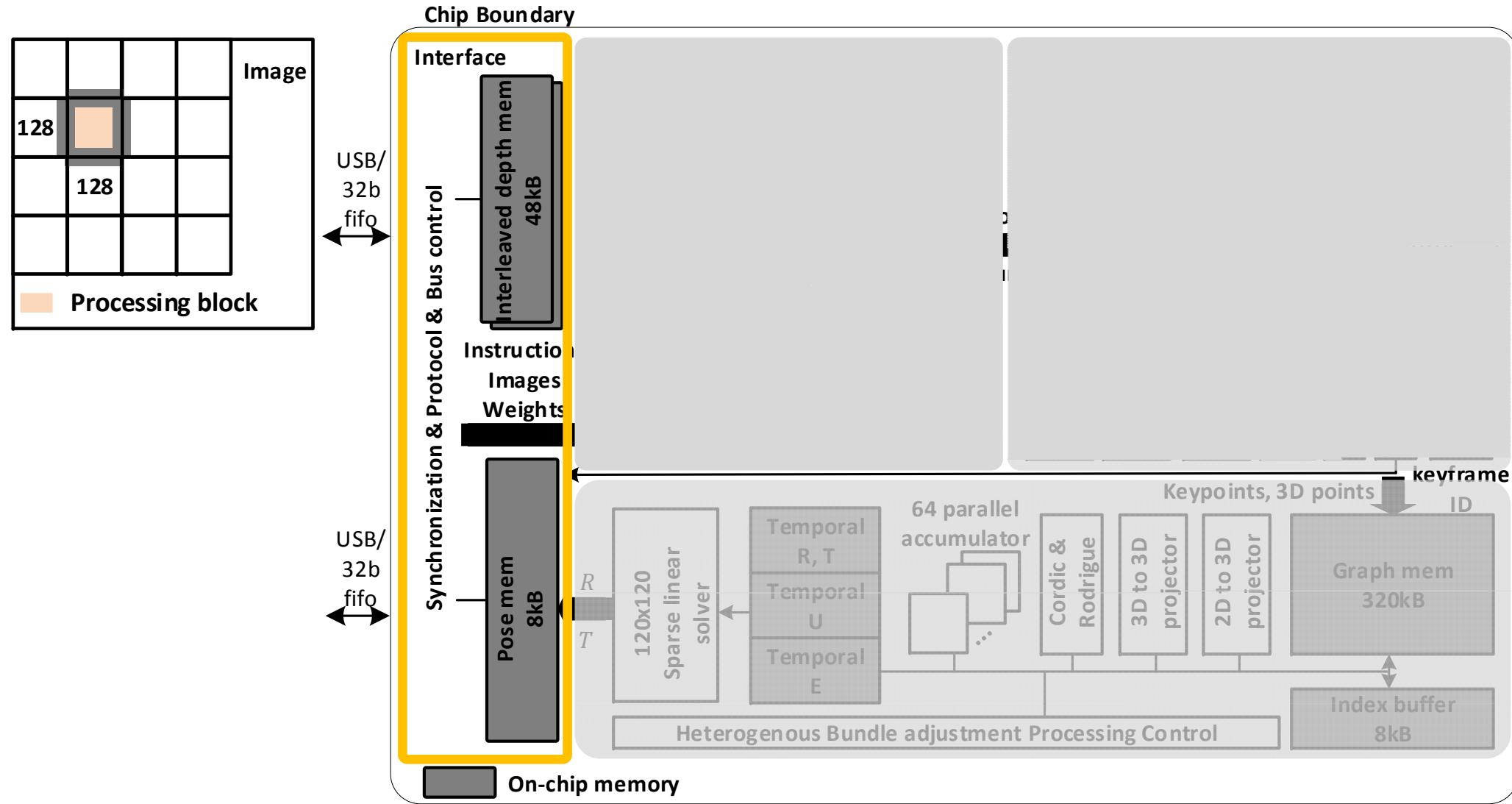
System	GPU (TITAN X)	CPU (GTX870M)	GPU (TK1)	ODROID	Arndale	Proposed
Performance @ VGA	135 fps	96 fps	22 fps	5.5 fps	4.2 fps	80 fps
Power	~400W	~100W	~20W	~10W	~5W	~240mW
Energy per frame	~2.9J	~1J	~0.9J	~1.8J	~1.1J	~3mJ
>External DRAM	Yes	yes	yes	yes	yes	No

Design and Optimizations

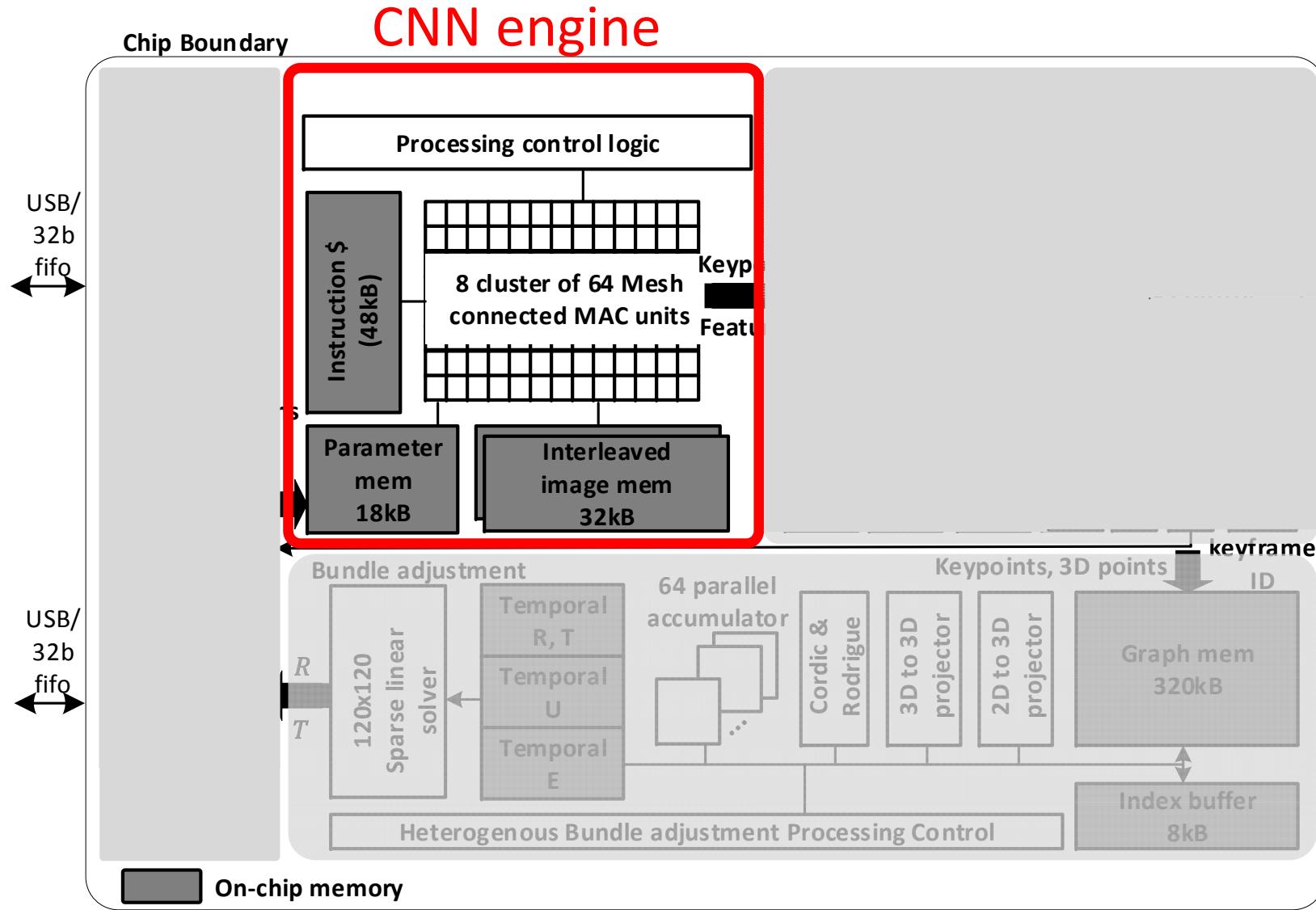
High-level Chip Architecture



High-level Chip Architecture



High-level Chip Architecture

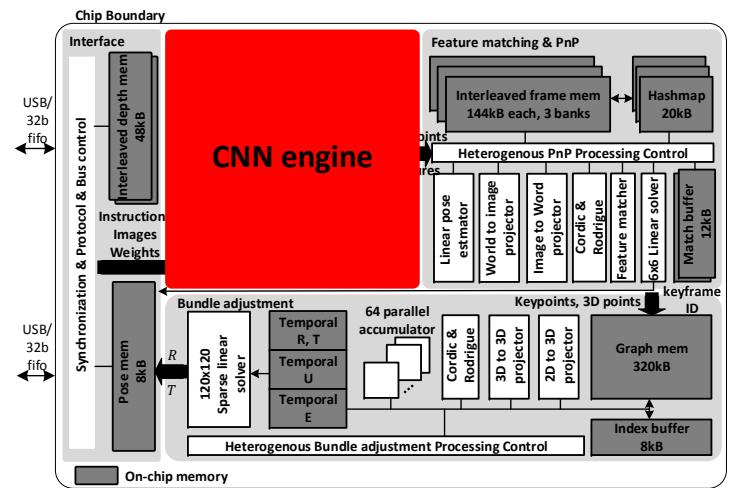


CNN-based Feature Matching

- Leverage state-of-the-art neural network
 - Memory & computation intensive
 - CNN feature; SIFT feature; ORB feature
 - 18% Better matching accuracy compared with SIFT
- Use a 4 layer triplet network [E Hoffer, 2016] for feature description
 - Trained on Multi-view Stereo Correspondence (MVS) dataset

Feature extraction network

layer	1	2	3	fc
convolution	3x3x16, Stride 2	3x3x32, Stride 2	3x3x64, Stride 2	64
Pooling	2x2	2x2	4x4	~
Nonlinear	Relu	Relu	Relu	~

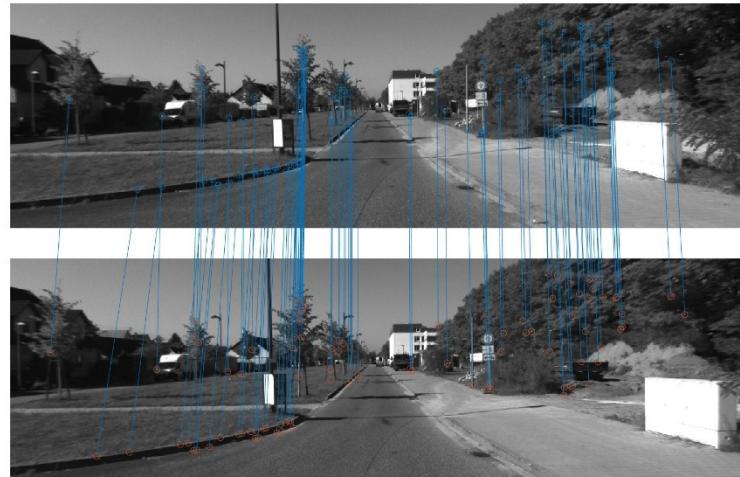


CNN-based Feature Matching

- Visualization under KITTI automotive benchmark
 - 18% Better matching accuracy compared with SIFT
 - ~0.5% degradation after quantization
 - 8 bit weight & 24 bit activation



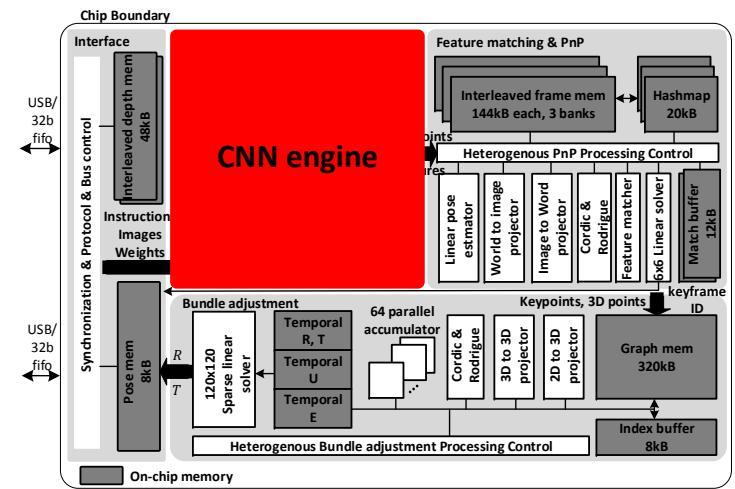
SIFT match



CNN match

Feature extraction accuracy

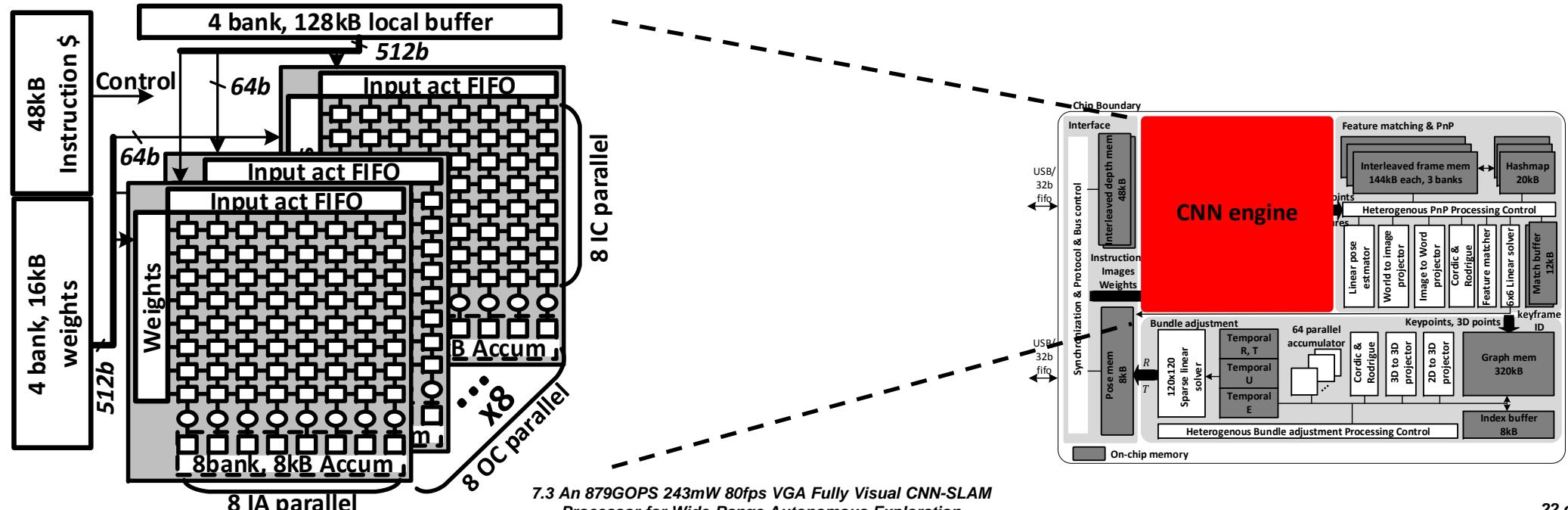
	FDR%
SIFT	27.29
CNN triplet (floating point)	22.39
CNN triplet (fixed point)	22.79



7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration

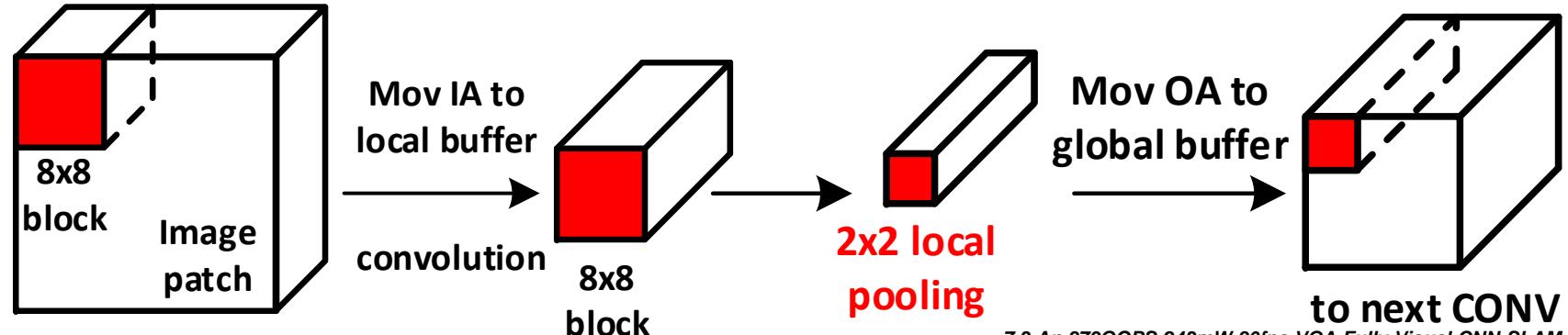
Feature Detection / Description Engine

- Runs every frame & every pixel → massive compute, dominates power
- Massive parallelism, data reuse & caching
 - 8x8x8 MAC units to process 8 IAs, 8 ICs & 8 OCs in parallel → mixed data reuse
 - Reconfigurable for DoG / extrema detection / pooling & CNN functions, etc
 - Cache weights with 1kB SRAM → 53% energy reduction

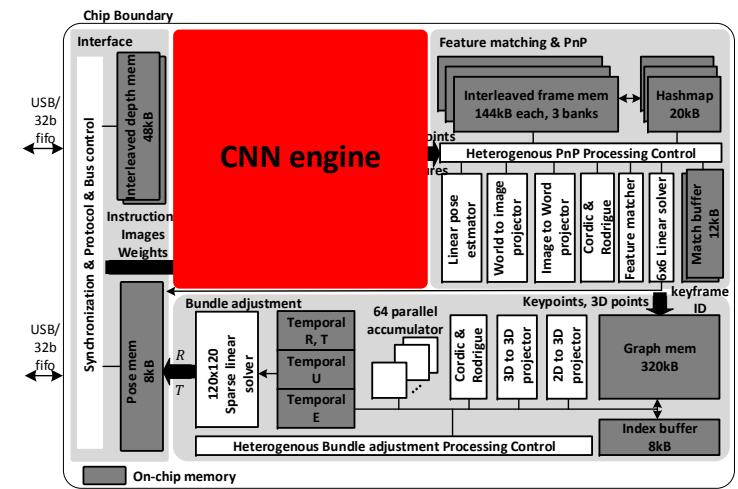


Feature Detection / Description Engine

- Massive parallelism, data reuse & caching
 - 8x8x8 MAC units to process 8 IAs, 8 ICs & 8 OCs in parallel → mixed data reuse
 - Reconfigurable for DoG / extrema detection / pooling & CNN functions, etc
 - Cache weights with 1kB SRAM
 - ~53% energy reduction on weight access
- Cross layer CNN processing → remove buffering between layers
 - Use 2x2/4x4 maxpool instead of 3x3

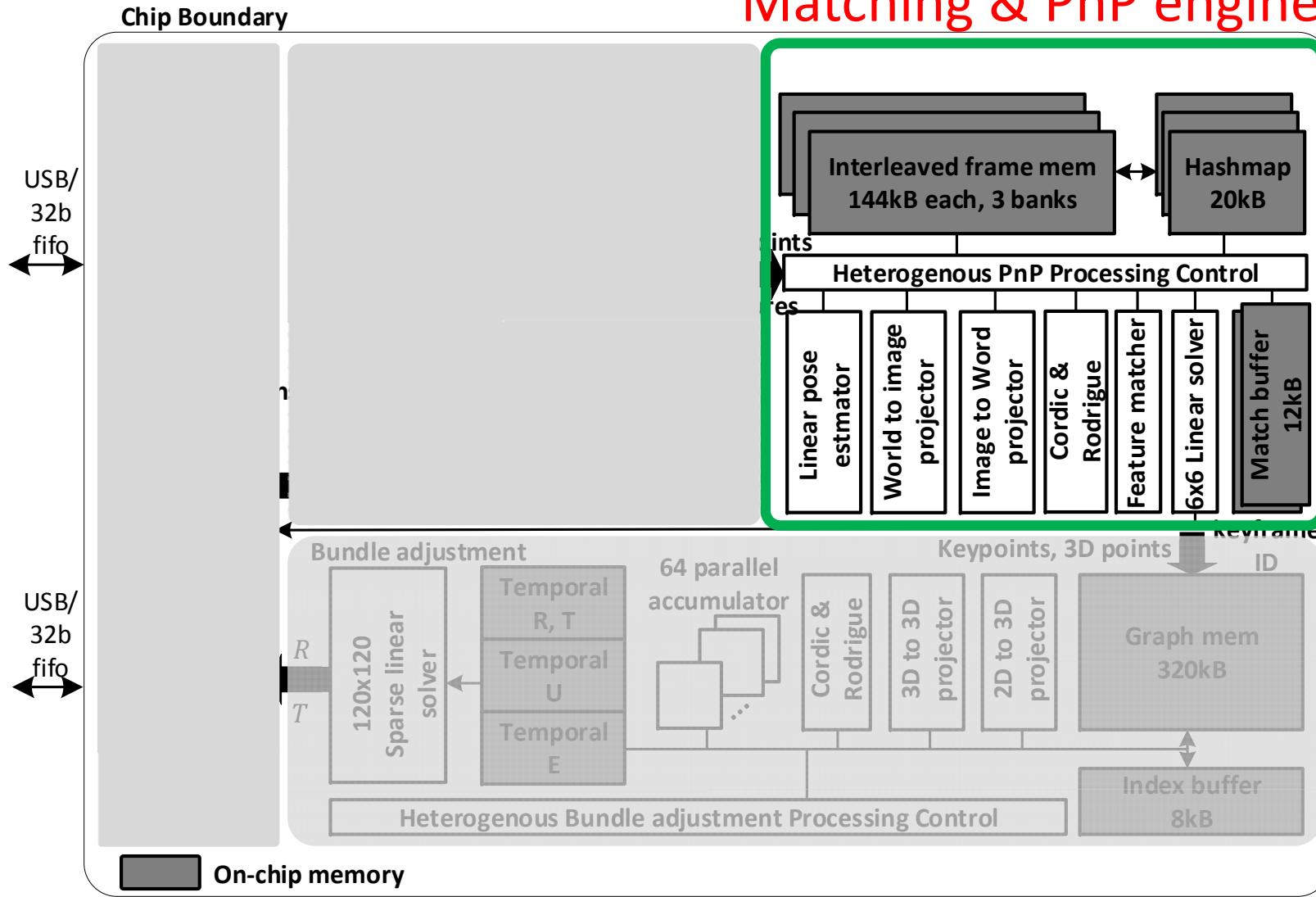


7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration



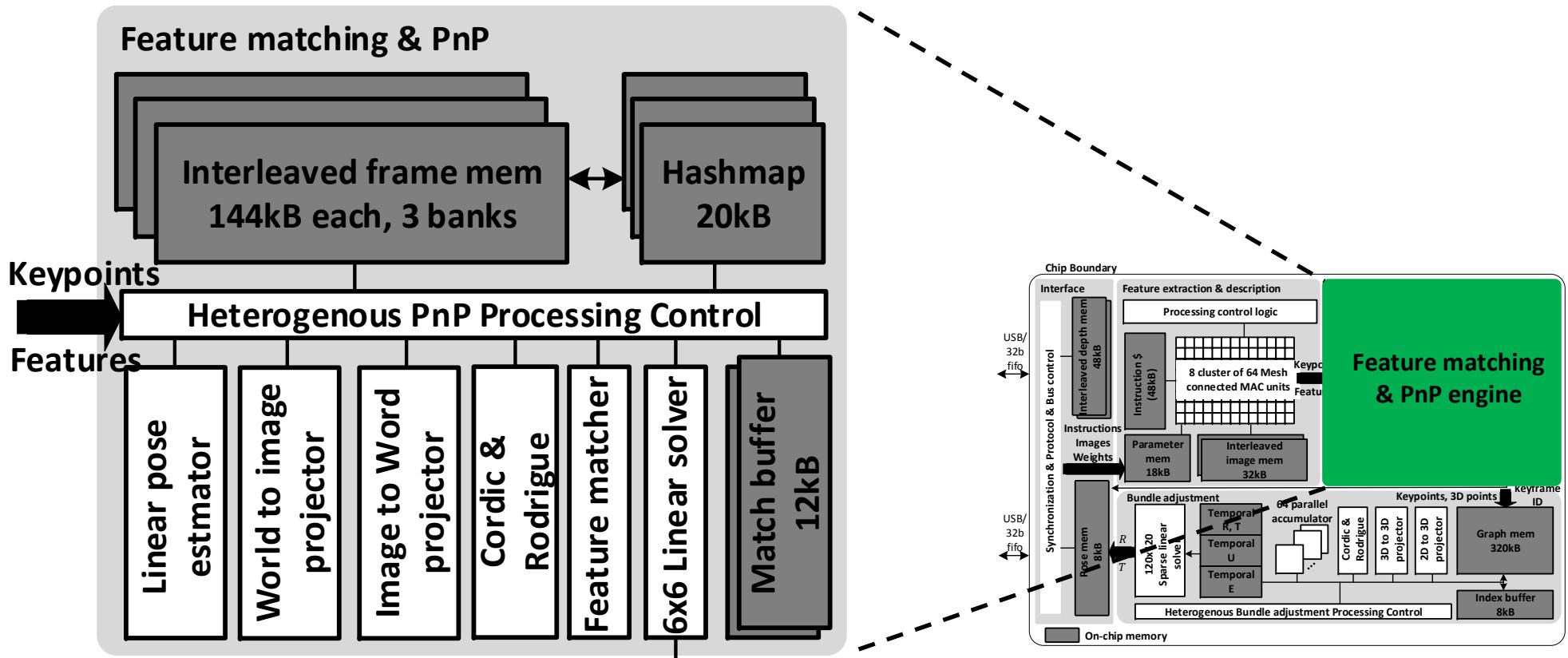
High-level Chip Architecture

Matching & PnP engine



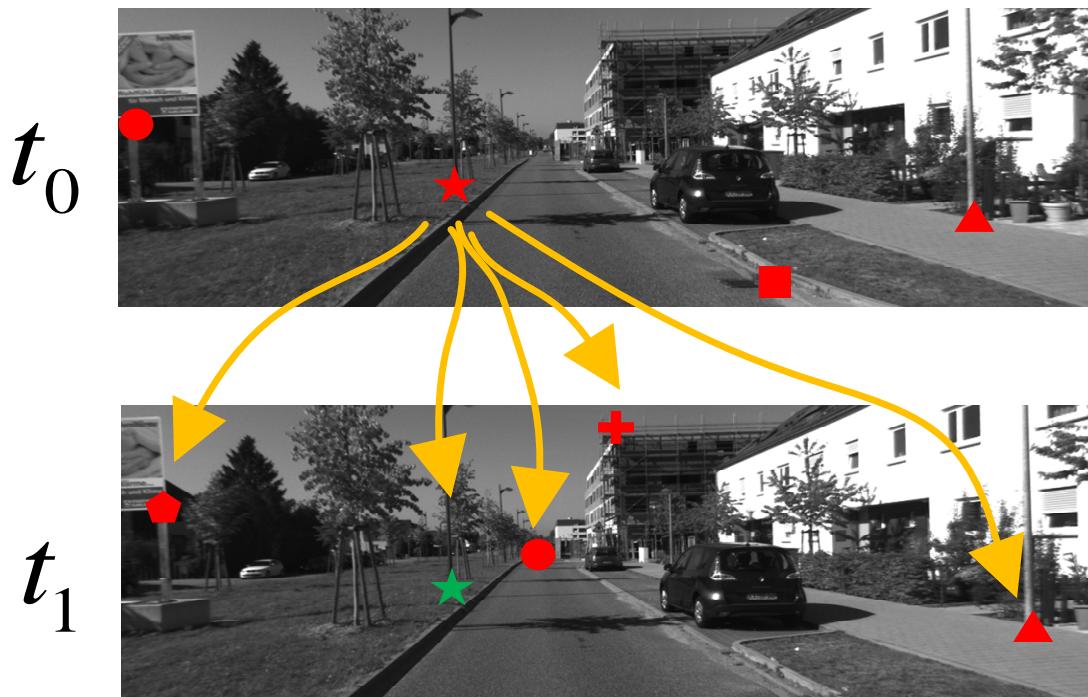
Feature Matching & PnP Engine

- Match features and solve for pose
 - Complex FSM with various kernels → runs at 80fps, 215MHz
 - 32 bit fixed point arithmetic

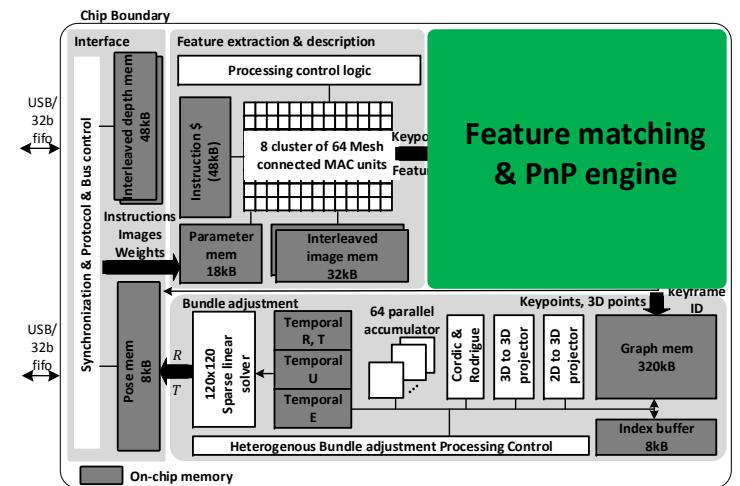


Feature Matching & PnP Engine

- >1000 CNN feature per frame
 - → bruteforce matching (1000x1000 complexity) is extremely costly

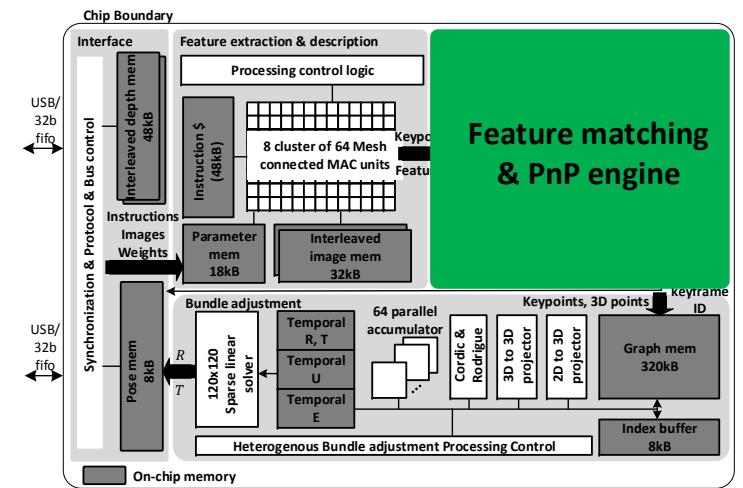
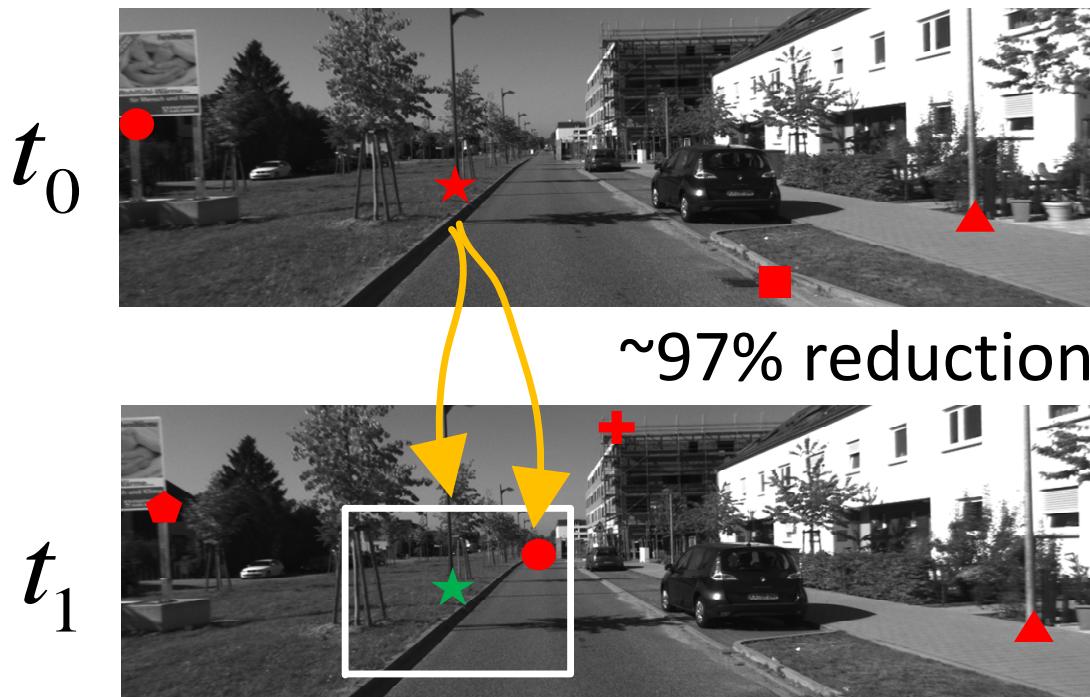


7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration



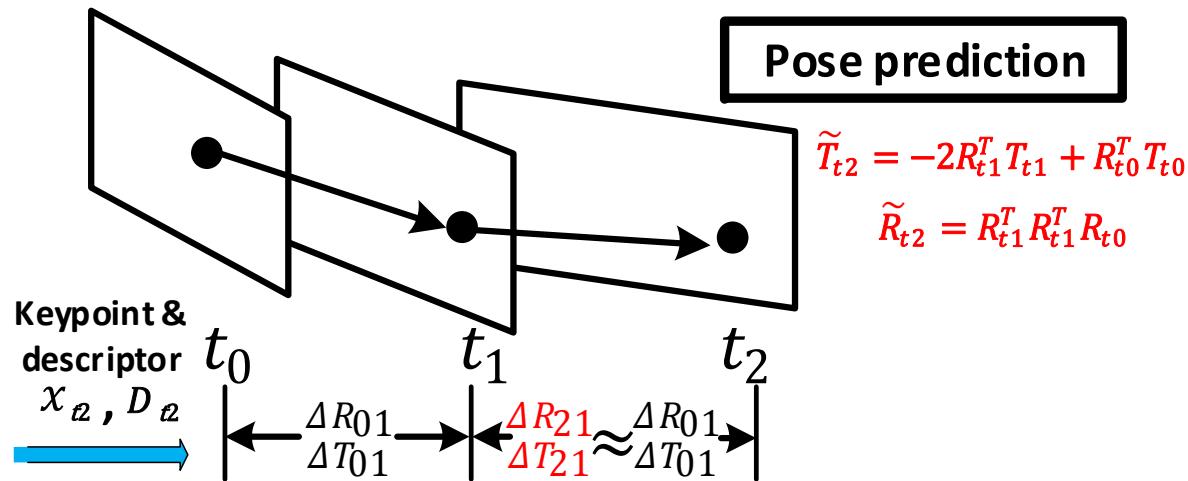
Feature Matching & PnP Engine

- >1000 CNN feature per frame
 - → bruteforce matching (1000x1000 complexity) is extremely costly
- Aggressively prune matching space to reduce complexity
 - → predict matching position and only match near the estimated location

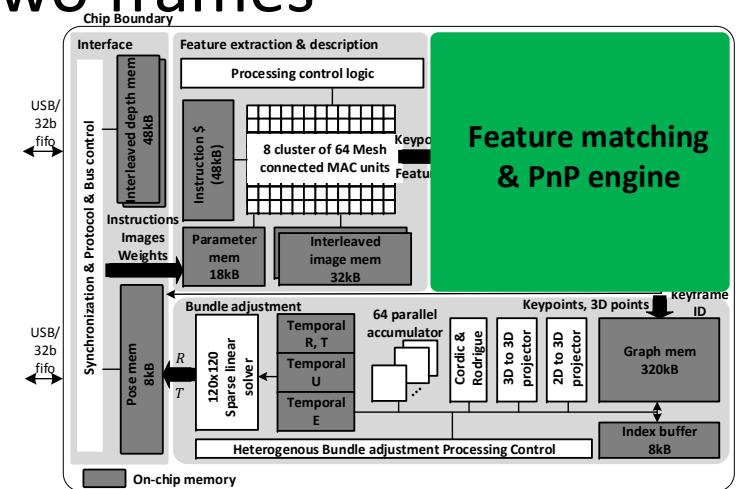


7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration

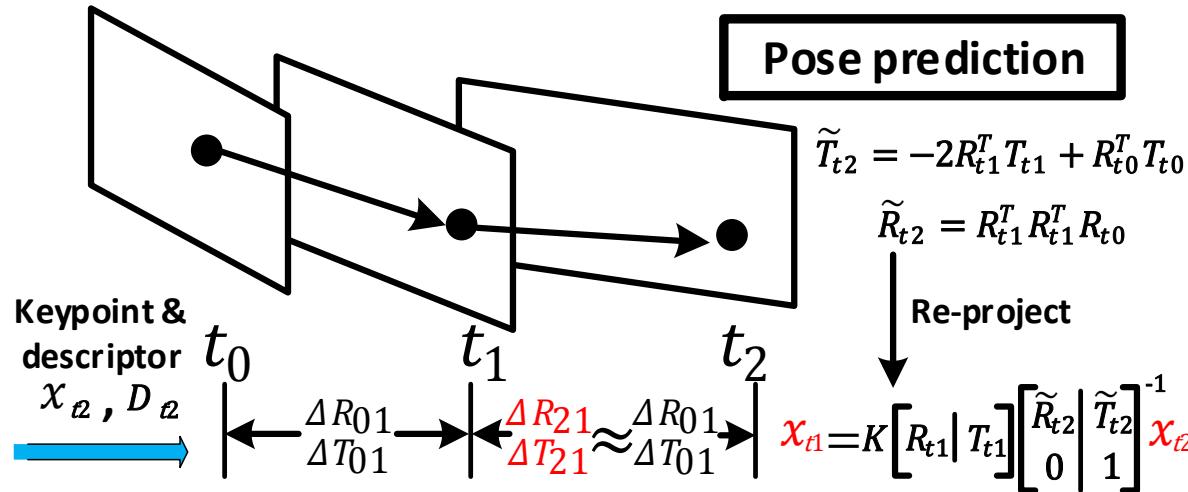
Feature Matching & PnP Engine



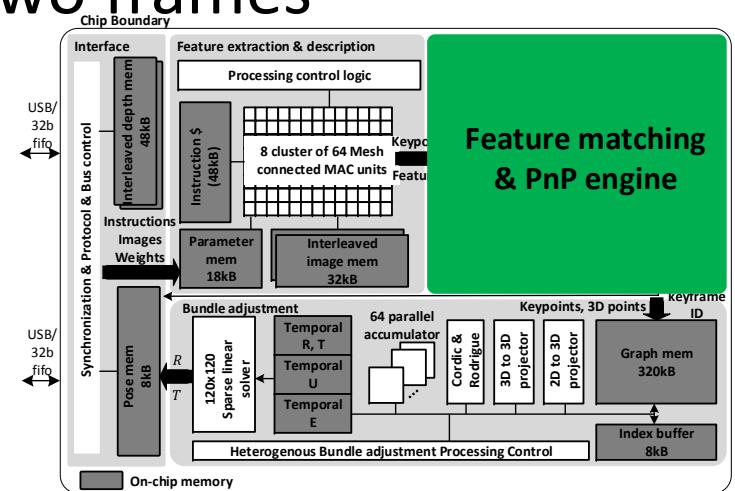
- Predict new pose from the poses of the previous two frames
 - Linear camera movement model



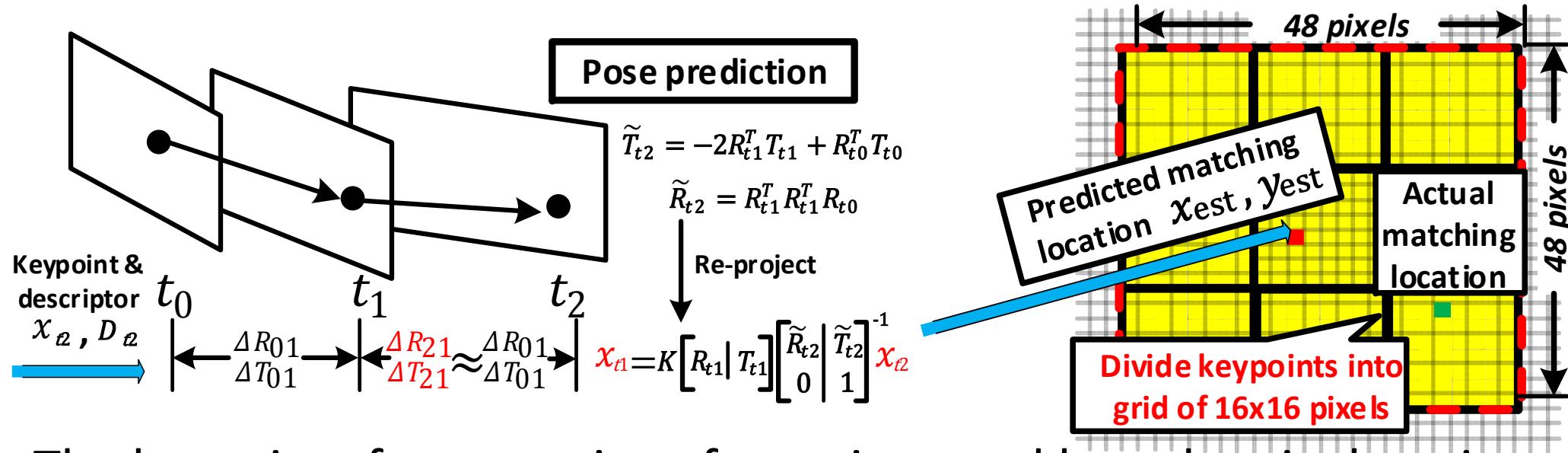
Feature Matching & PnP Engine



- Predict new pose from the poses of the previous two frames
 - Linear camera movement model
 - Predict the matching pixel onto the previous frame



Feature Matching & PnP Engine

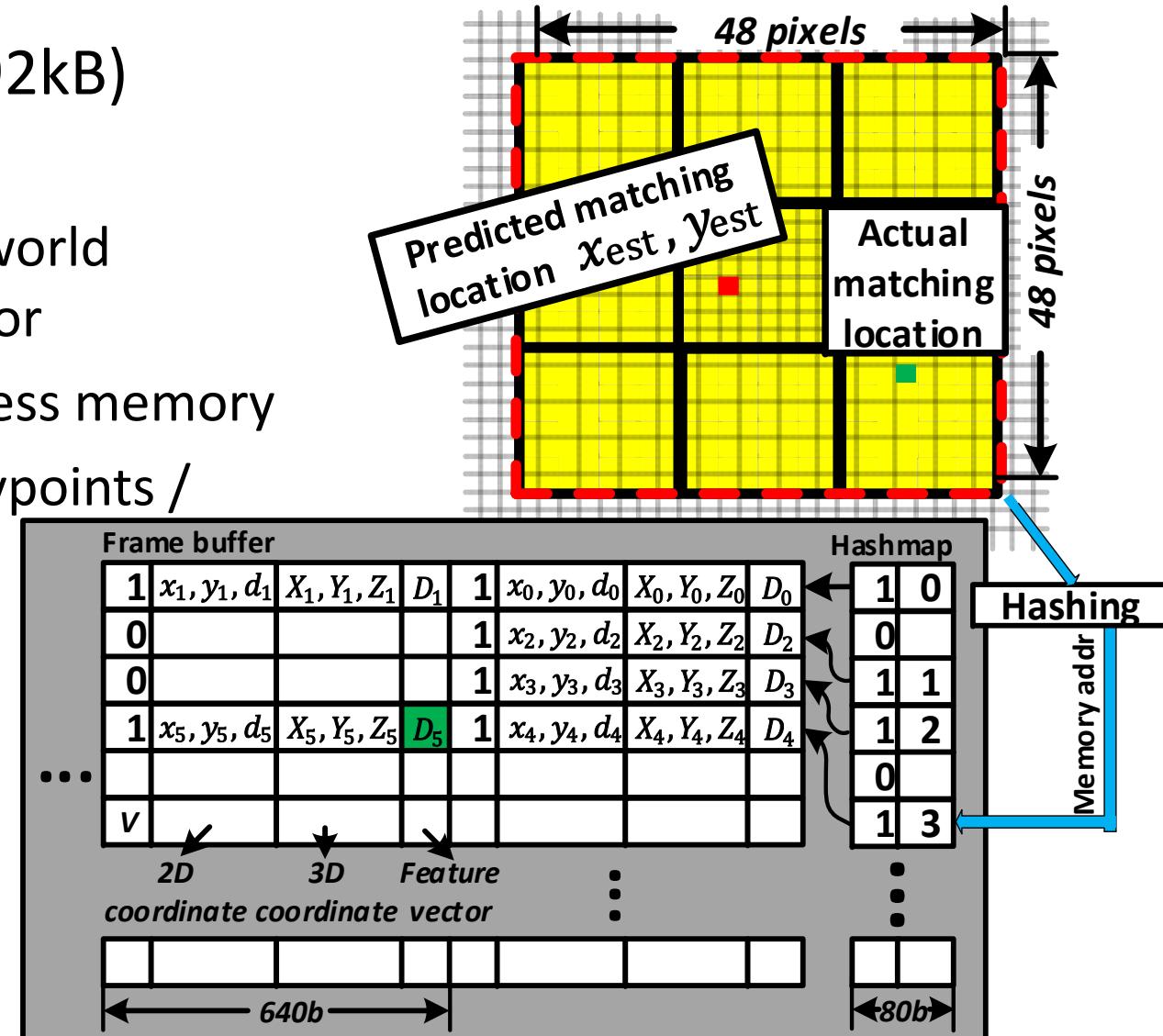


- The keypoints from previous frame is stored based on its location
 - Partition the image into 16x16 grid
 - Search the center block with 8 neighboring blocks $\rightarrow \sim \pm 24 \times \pm 24$ search window
 - Eliminate 97% of unnecessary matchings without accuracy degradation (<0.1%).

 Grouped memory word of 16x16 pixels

Feature Matching & PnP Engine

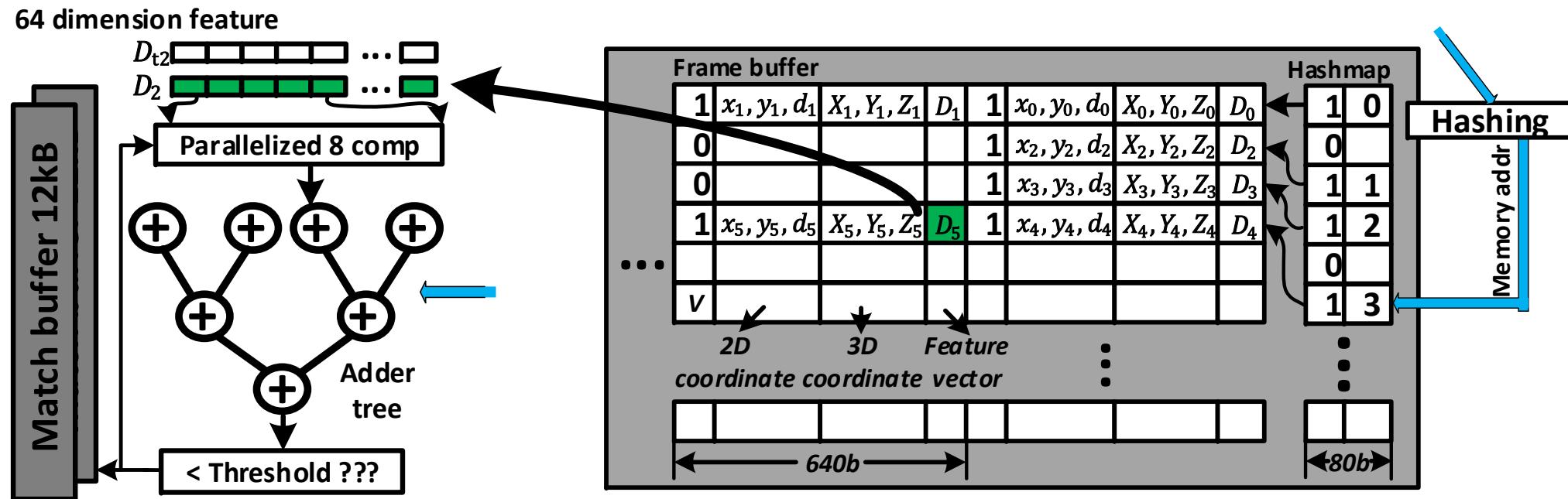
- Hierarchical frame buffer (492kB)
 - Sparse populated hash table
 - Stores 2D keypoints & 3D world coordinates & feature vector
 - Hash the block address to access memory
 - Each 16x16 grid will hold 8 keypoints / features at maximum



7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration

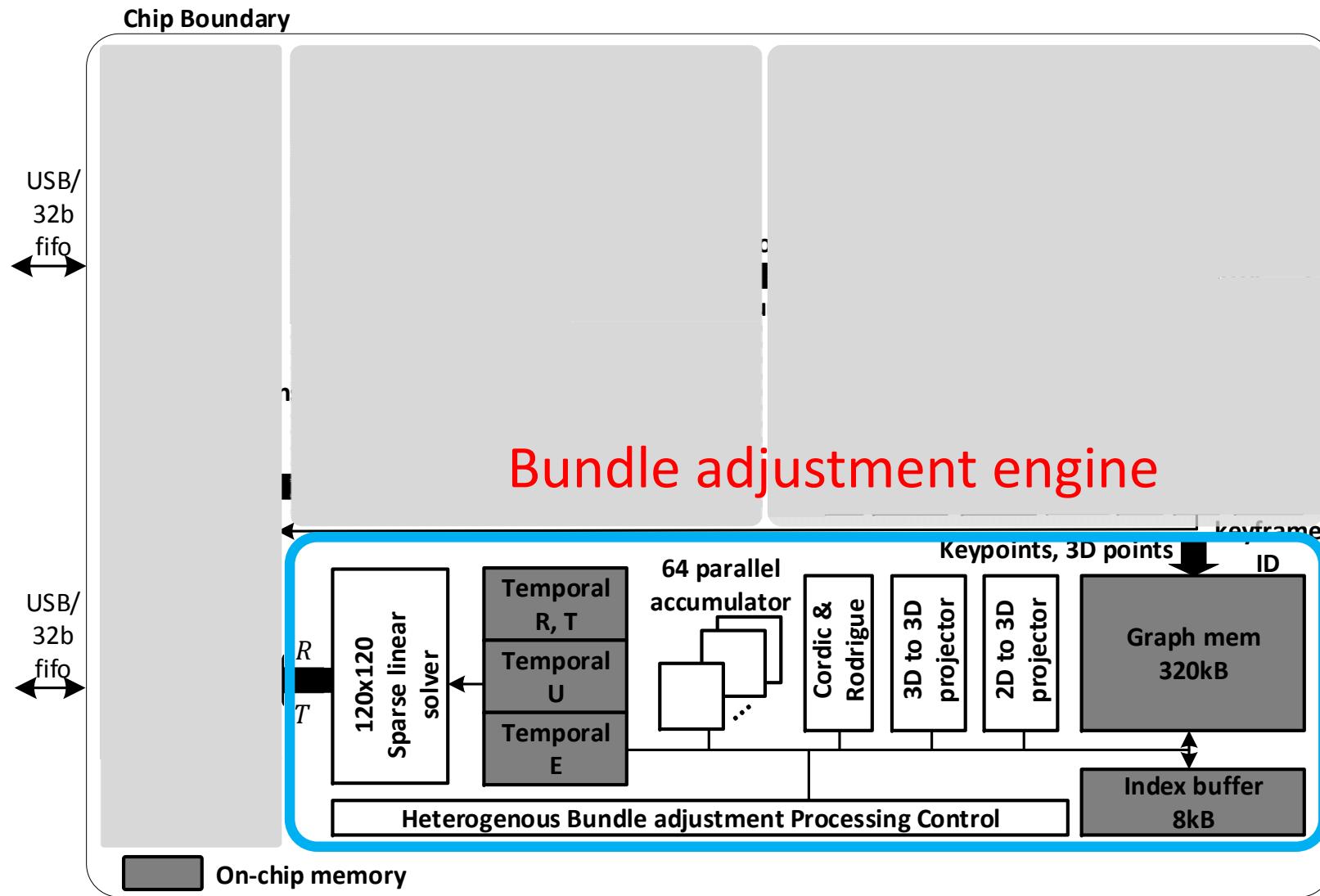
Feature Matching & PnP Engine

- Feature matching is performed based on L2 norm
 - Each feature vector has 64 dimension
 - Processed with 8 parallel units → taking up to 8 cycles
- Terminate matching if already mismatch in early stages
 - → ~4.8 average cycles



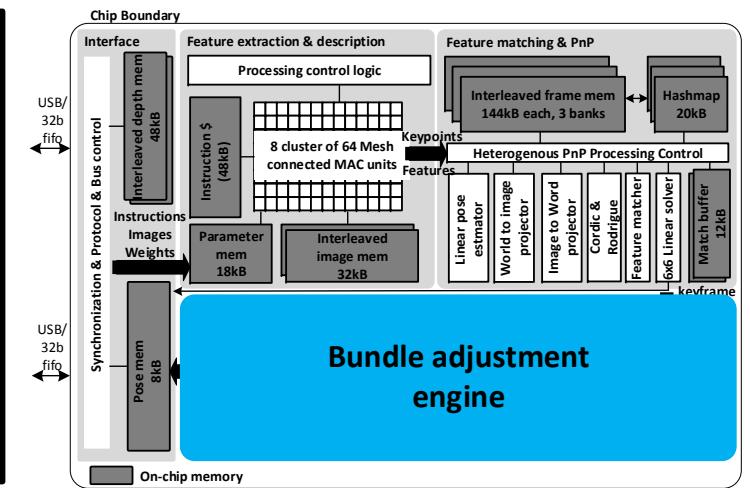
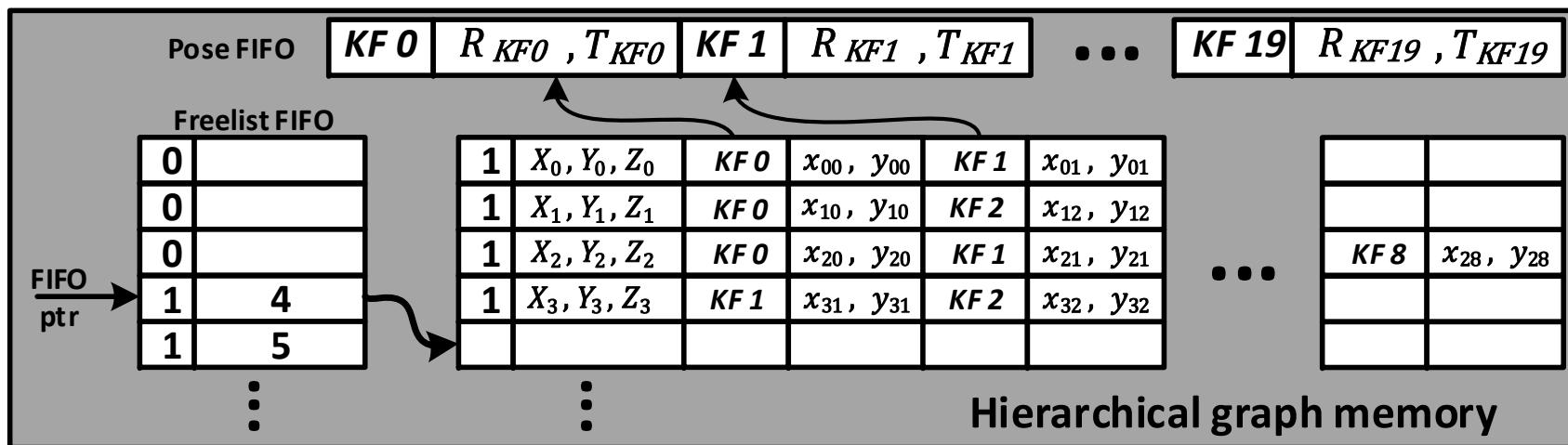
7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration

High-level Chip Architecture



Graph Memory

- Hierarchical graph memory (320kB)
 - Stores 20 keyframes, 4096 keypoints at maximum
 - A separate FIFO serves to eliminate/insert a keyframe into the graph memory
 - Matched keypoints are merged into a single entry



BA & Fixed Point Implementation

- Bundle adjustment is solved numerically with LM algorithm
 - Computing Jacobian numerically needs to apply very small increments on pose
 - Extremely large dynamic range and high precision is required
 - Prior works [VLSI'18, JSSC'15] uses 64 bit floating point precision

Re-projection with
small increment

$$\begin{aligned} \mathbf{x}_i &= \begin{bmatrix} R & T \end{bmatrix} X_i \\ \mathbf{x}'_i &= \begin{bmatrix} R + \Delta R & T + \Delta T \end{bmatrix} X_i \end{aligned}$$

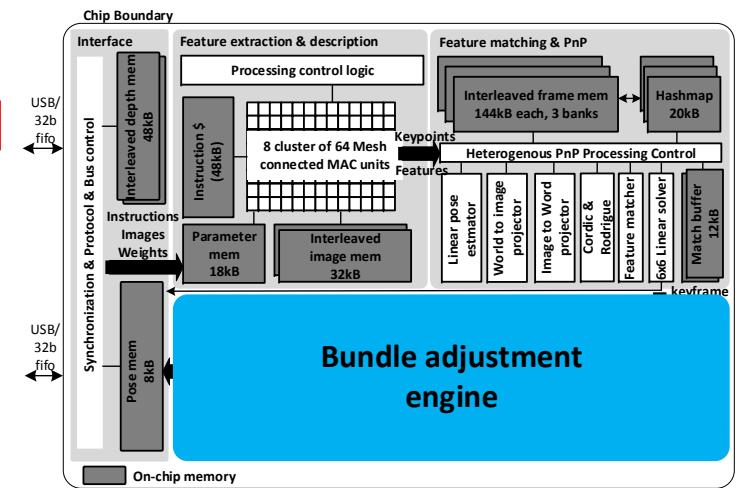
Jacobian

$$J = \frac{\mathbf{x}'_X}{\|\mathbf{x}'_Z\|} - \frac{\mathbf{x}_X}{\|\mathbf{x}_Z\|}$$

High precision required

Levenberg-Marquart
iteration

$$(J^T J + \lambda I) \Delta x = -J^T \Delta \varepsilon$$



BA & Fixed Point Implementation

- Bundle adjustment is solved numerically with LM algorithm
 - Subtract common offset before normalizing the projected 2D points into homogeneous coordinates
 - Reduce precision → use 32 bit fixed point → ~40% energy reduction

Re-projection with small increment

$$\begin{aligned} \mathbf{x}_i &= \begin{bmatrix} R & T \end{bmatrix} X_i \\ \mathbf{x}'_i &= \begin{bmatrix} R + \Delta R & T + \Delta T \end{bmatrix} X_i \end{aligned}$$

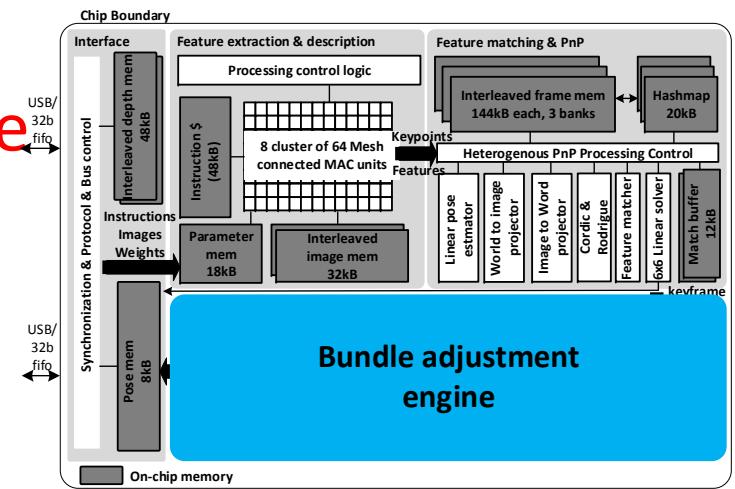
Jacobian

$$J = \frac{\mathbf{x}'_x \mathbf{x}_z - \mathbf{x}_x \mathbf{x}'_z}{\|\mathbf{x}'_x \mathbf{x}_z\|}$$

Pre-condition the range

Levenberg-Marquart iteration

$$(J^T J + \lambda I) \Delta x = -J^T \Delta \varepsilon$$



BA & Matrix Solver

- Bundle adjustment is solved numerically with LM algorithm
 - After Jacobian is computed for all keypoints & keyframes
 - Linearize the problem → sparse 120x120 matrix with diagonal sub-matrices
 - Solve 6x6 matrices sequentially

Re-projection with small increment

$$\begin{aligned} \mathbf{x}_i &= \begin{bmatrix} R & T \end{bmatrix} X_i \\ \mathbf{x}'_i &= \begin{bmatrix} R + \Delta R & T + \Delta T \end{bmatrix} X_i \end{aligned}$$

Jacobian

$$J = \frac{\mathbf{x}'_x \mathbf{x}_z - \mathbf{x}_x \mathbf{x}'_z}{\|\mathbf{x}'_x \mathbf{x}_z\|}$$

Levenberg-Marquart iteration

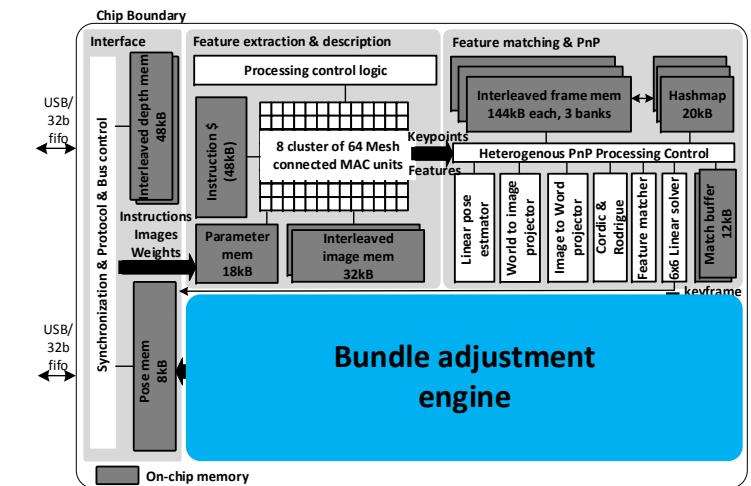
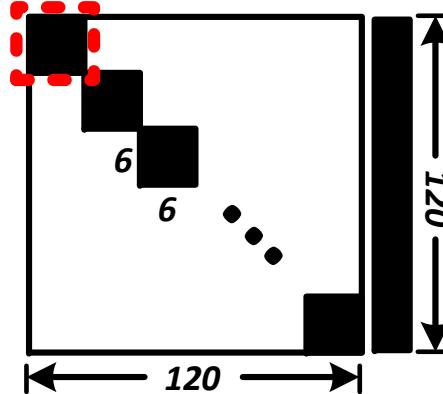
$$(J^T J + \lambda I) \Delta x = -J^T \Delta \varepsilon$$

Linearize

$$H \Delta x = \varepsilon$$

Solve 6x6 matrix sequentially

Hessian matrix

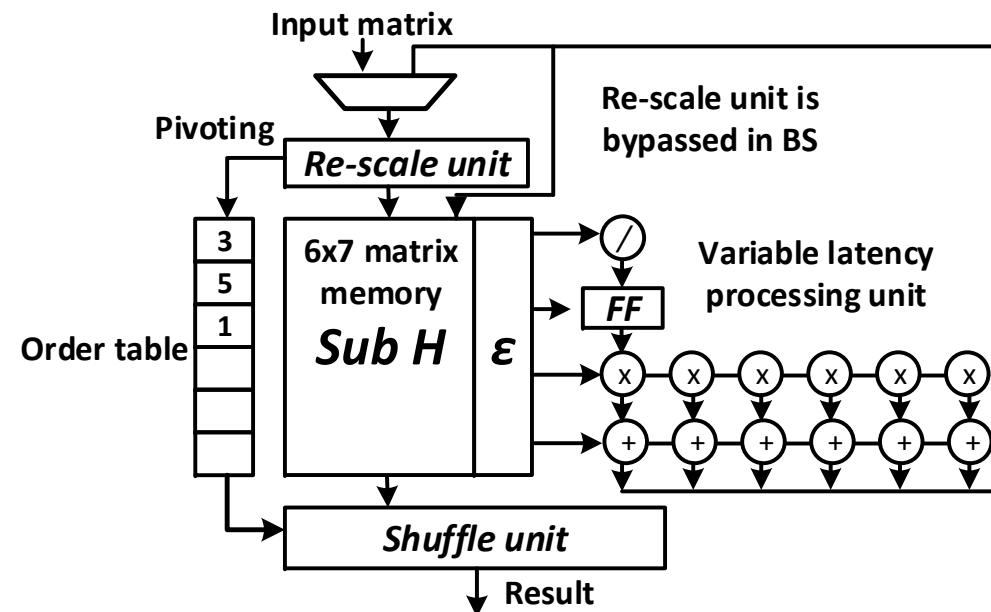


BA & Fixed Point Implementation

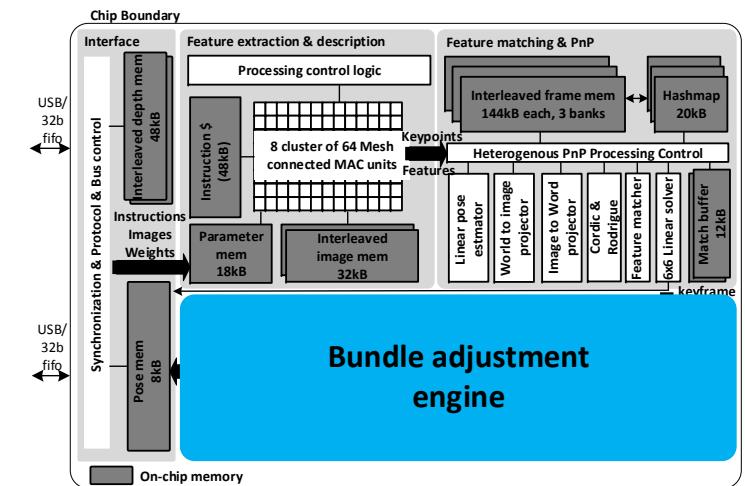
- Numerically stable matrix solver
 - Gaussian elimination and back substitution
 - Pivoting → improve numerically stability with 32bit fixed point
 - Shared parallel arithmetic units between GE and BS

$$\text{Gaussian elimination} \quad T\Delta x = \varepsilon$$

$$\text{Back substitution} \quad \Delta x$$

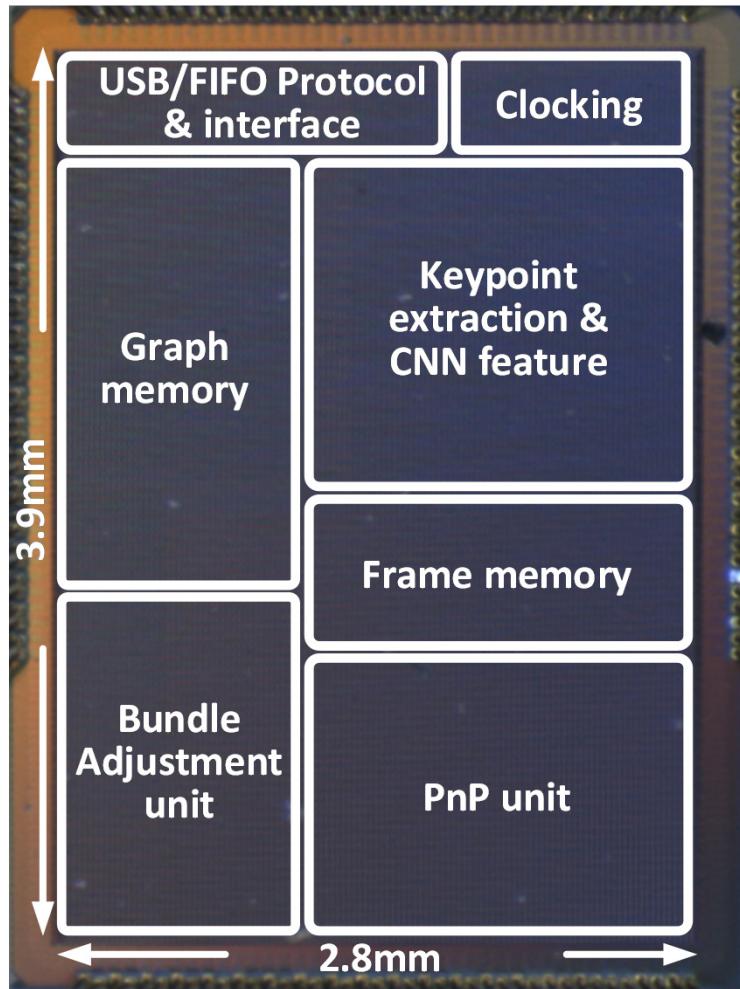


7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration



Measurements

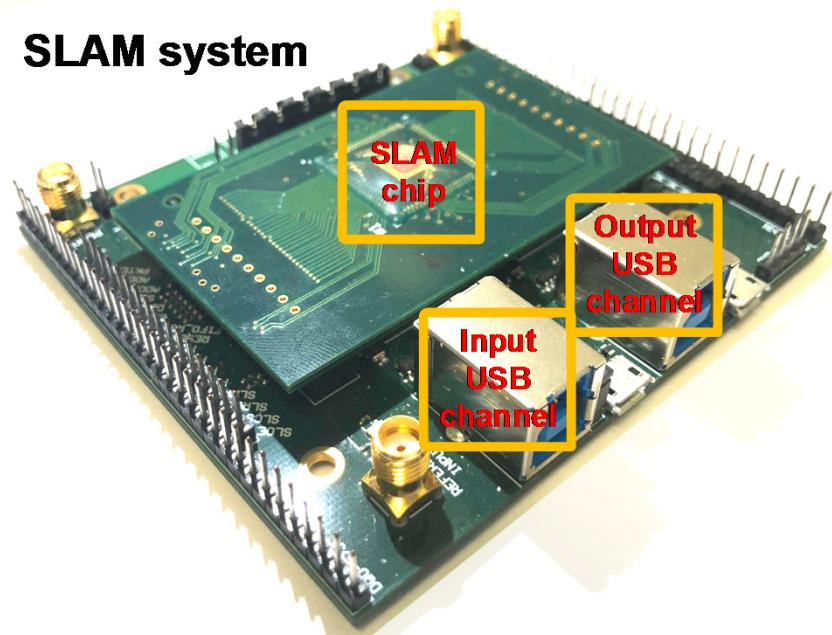
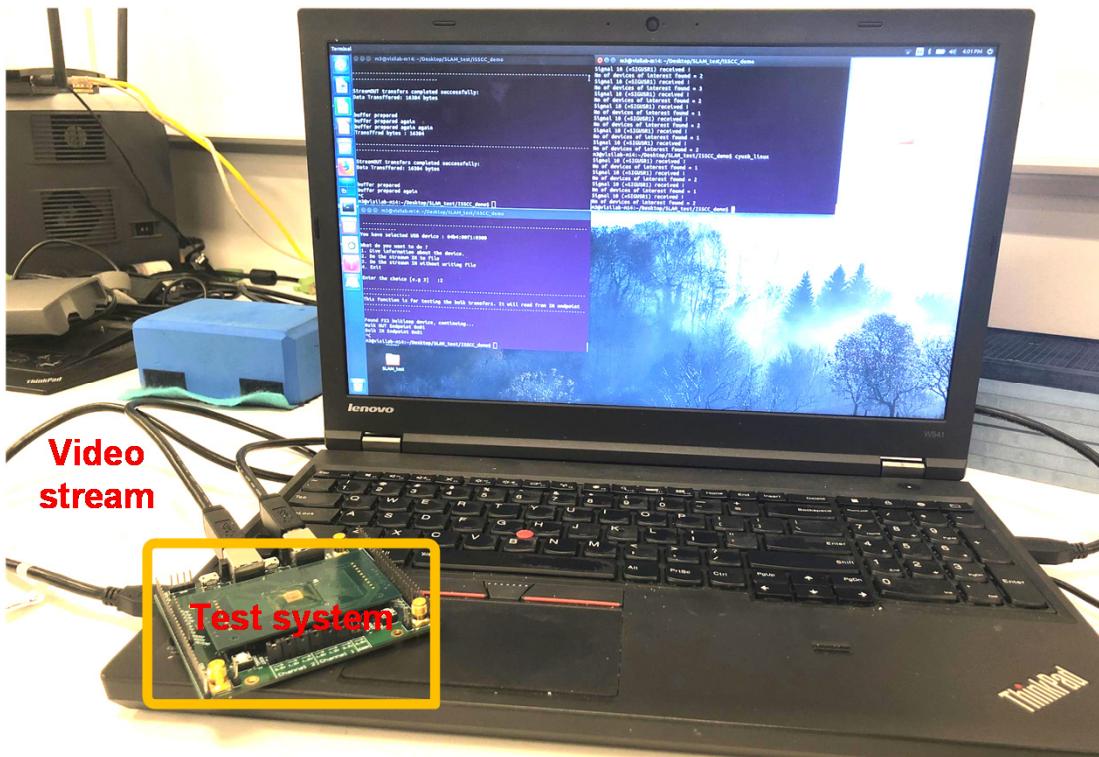
Die Photo & Performance Summary



	This work
Vision frontend	CNN feature
Pose tracking	PnP
Graph optimization	Local BA
Technology	28nm
Chip area	10.92mm ²
On-chip memory	1126kB
Frequency	215MHz
Throughput & image size	640 X 480, 80fps
Operation range	± 500m
Track points / frame	1000
Operating voltage	0.9V
Performance	879.6 GOPS @ 0.9V, 215MHz 329.8 GOPS @ 0.63V, 90MHz
Power	243.6 mW @ 0.9V 61.75 mW @ 0.63V
Energy efficiency	3.6 TOPS/W @ 0.9V 5.34 TOPS/W @ 0.63V

Chip Measurement & Demo Setup

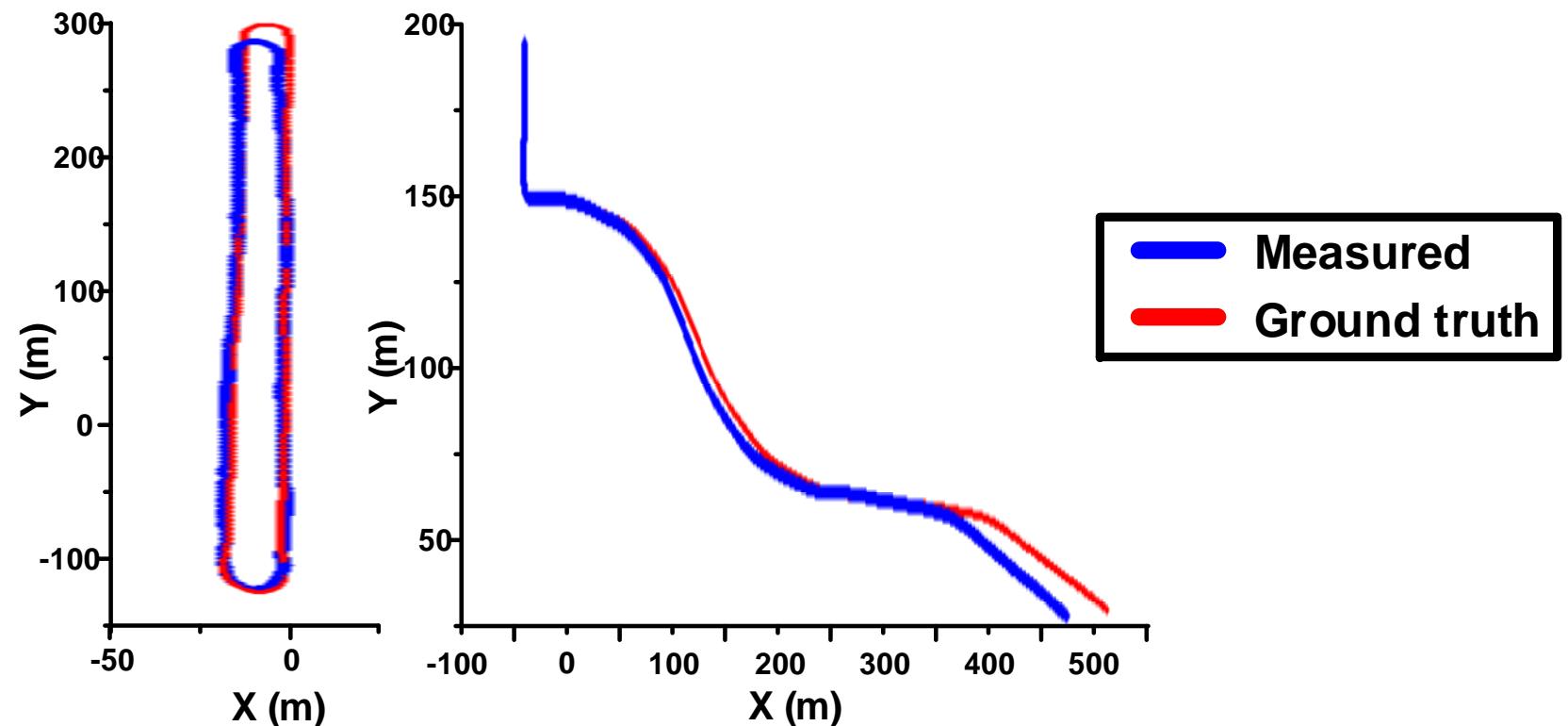
- Input/output streams to/from the chip via a USB3.0 interface
 - Image streams & controls
 - USB3.0 signals are converted to parallel on boards with Cypress FX3 chip



7.3 An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration

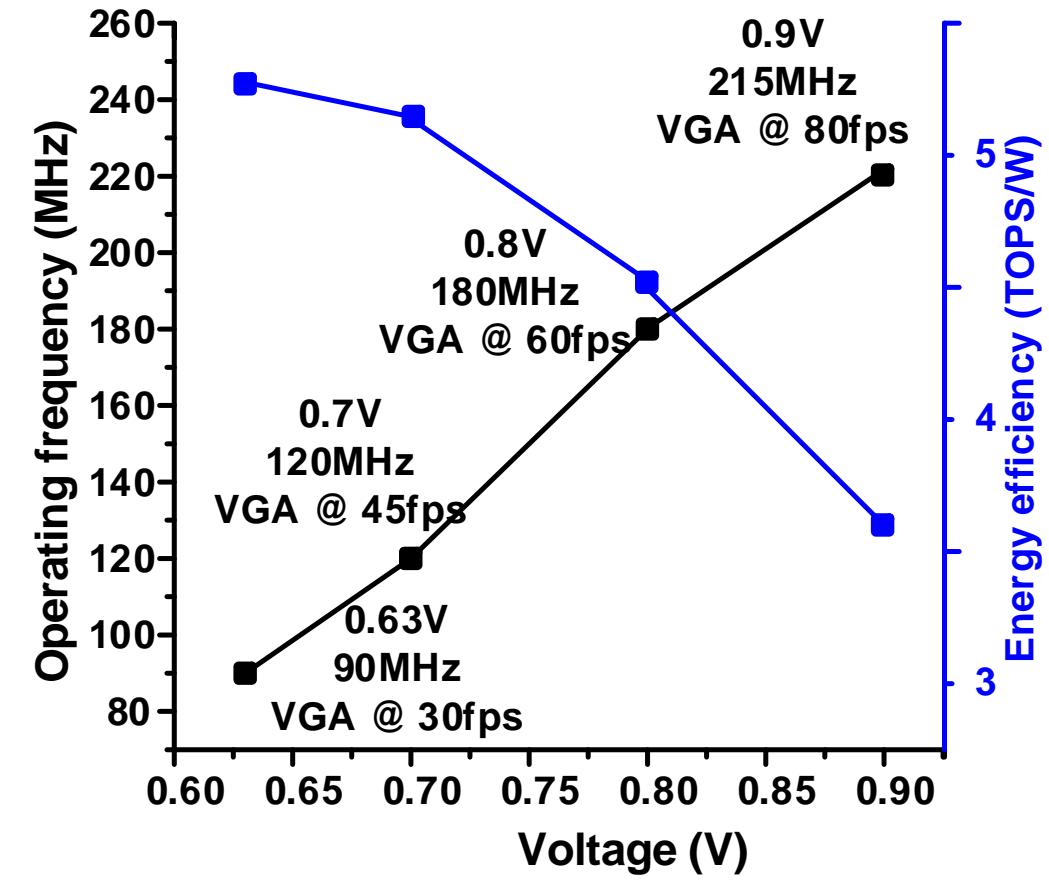
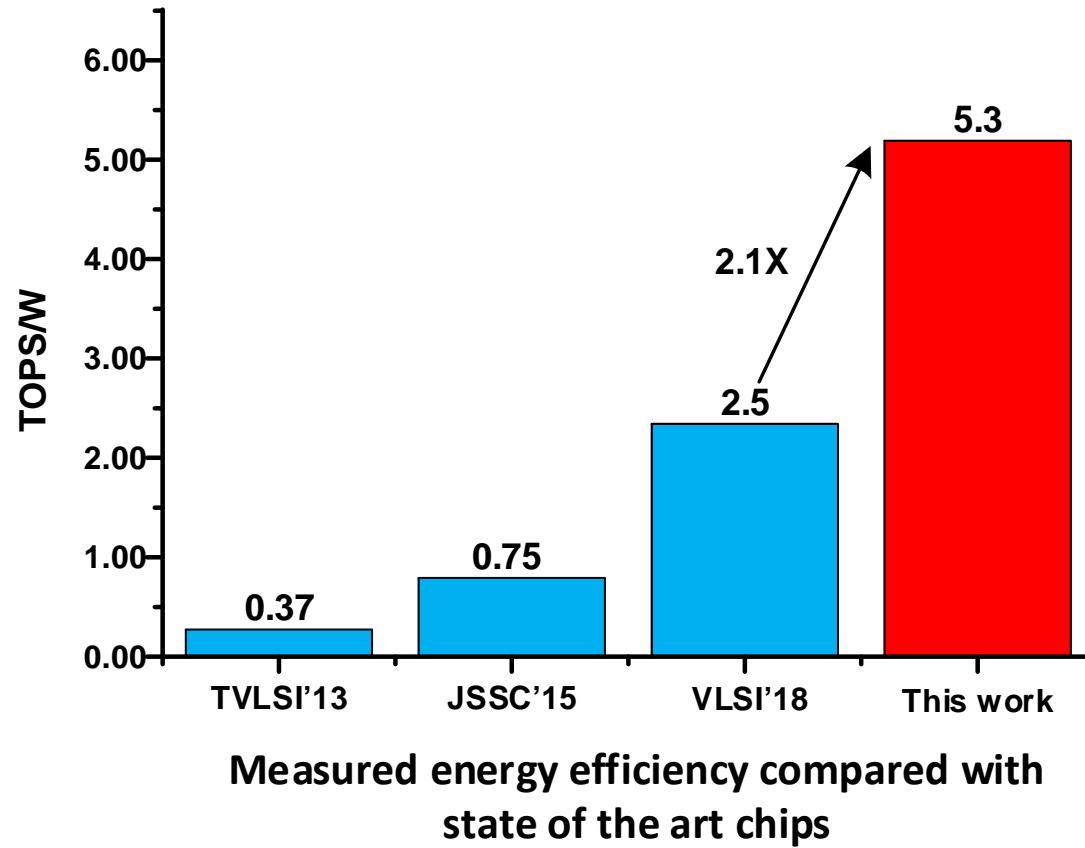
Chip Measurements

- Chip is tested with standard KITTI sequences, each consists >1000 images
 - Industrial standard automotive dataset
 - ~0.5% rotation error and ~2% translation error



Performance Scaling

- Performance on various throughput and resolution



Performance Comparison

		This work	TVLSI 2013[3]	JSSC 2015[4]	VLSI 2018[5]
Complete Visual SLAM system	Vision frontend	✓ CNN feature	✓ Marker recognition	✗	✓ Harris feature
	Pose tracking	✓ Visual	✗	✓ Visual	✗ Inertial sensor
	Graph optimization	✓ Local BA	✗	✗	✓ Local BA
Technology	28nm	180nm	65nm	65nm	
Chip area	10.92mm ²	28.75mm ²	3.25mm ²	20mm ²	
On-chip memory	1126kB	N.A	96KB	854kB	
Frequency	215MHz	200MHz	133MHz	83.3MHz	
Throughput & image size	640 X 480, 80fps	640 X 480, 100fps	640 X 480, 95fps	752 X 480, 90fps	
Operation range	± 500m	Not reported			
Track points / frame	1000	9	33	200	
Operating voltage	0.63-0.9V	1.8V	1.2V	1.2V	
Performance	879.6 GOPS @ 0.9V, 215MHz 329.8 GOPS @ 0.63V, 90MHz	153.6 GOPS	20.2 GOPS	10.5-59.1 GOPS	
Power	243.6 mW @ 0.9V, 215MHz 61.75 mW @ 0.63V, 90MHz	413 mW	27 mW	24 mW* (excluding IMU power)	
Energy efficiency	3.6 TOPS/W @ 0.9V, 215MHz 5.34 TOPS/W @ 0.63V, 90MHz	0.37 TOPS/W	0.75 TOPS/W	0.43-2.5 TOPS/W	

Performance Comparison

		This work	TVLSI 2013[3]	JSSC 2015[4]	VLSI 2018[5]
Complete Visual SLAM system	Vision frontend	✓ CNN feature	✓ Marker recognition	✗	✓ Harris feature
	Pose tracking	✓ Visual	✗	✓ Visual	✗ Inertial sensor
	Graph optimization	✓ Local BA	✗	✗	✓ Local BA
Technology	28nm	180nm	65nm	65nm	
Chip area	10.92mm ²	28.75mm ²	3.25mm ²	20mm ²	
On-chip memory	1126kB	N.A	96KB	854kB	
Frequency	215MHz	200MHz	133MHz	83.3MHz	
Throughput & image size	640 X 480, 80fps	640 X 480, 100fps	640 X 480, 95fps	752 X 480, 90fps	
Operation range	± 500m	Not reported due to limited range			
Track points / frame	1000	9	33	200	
Operating voltage	0.63-0.9V	1.8V	1.2V	1.2V	
Performance	879.6 GOPS @ 0.9V, 215MHz 329.8 GOPS @ 0.63V, 90MHz	153.6 GOPS	20.2 GOPS	10.5-59.1 GOPS	
Power	243.6 mW @ 0.9V, 215MHz 61.75 mW @ 0.63V, 90MHz	413 mW	27 mW	24 mW* (excluding IMU power)	
Energy efficiency	3.6 TOPS/W @ 0.9V, 215MHz 5.34 TOPS/W @ 0.63V, 90MHz	0.37 TOPS/W	0.75 TOPS/W	0.43-2.5 TOPS/W	

Performance Comparison

		This work	TVLSI 2013[3]	JSSC 2015[4]	VLSI 2018[5]
Complete Visual SLAM system	Vision frontend	✓ CNN feature	✓ Marker recognition	✗	✓ Harris feature
	Pose tracking	✓ Visual	✗	✓ Visual	✗ Inertial sensor
	Graph optimization	✓ Local BA	✗	✗	✓ Local BA
Technology	28nm	180nm	65nm	65nm	
Chip area	10.92mm ²	28.75mm ²	3.25mm ²	20mm ²	
On-chip memory	1126kB	N.A	96KB	854kB	
Frequency	215MHz	200MHz	133MHz	83.3MHz	
Throughput & image size	640 X 480, 80fps	640 X 480, 100fps	640 X 480, 95fps	752 X 480, 90fps	
Operation range	± 500m	Not reported due to limited range			
Track points / frame	1000	9	33	200	
Operating voltage	0.63-0.9V	1.8V	1.2V	1.2V	
Performance	879.6 GOPS @ 0.9V, 215MHz 329.8 GOPS @ 0.63V, 90MHz	153.6 GOPS	20.2 GOPS	10.5-59.1 GOPS	
Power	243.6 mW @ 0.9V, 215MHz 61.75 mW @ 0.63V, 90MHz	413 mW	27 mW	24 mW* (excluding IMU power)	
Energy efficiency	3.6 TOPS/W @ 0.9V, 215MHz 5.34 TOPS/W @ 0.63V, 90MHz	0.37 TOPS/W	0.75 TOPS/W	0.43-2.5 TOPS/W	

Conclusion

- Single chip, energy efficiency CNN-SLAM processor that implements full visual SLAM pipeline
- Feature extraction with highly parallelized CNN-engine achieves 18% better accuracy
- Aggressively pruned feature matching with hierarchical memory operates with >1000 keypoints per frame at 80fps VGA
- Numerically stable 32 bit fixed-point implementation with hardware sharing of different kernels
- System built and demonstrated for real-time stream processing

Conclusion

- Single chip, energy efficiency CNN-SLAM processor that implements full visual SLAM pipeline
- Feature extraction with highly parallelized CNN-engine achieves 18% better accuracy
- Aggressively pruned feature matching with hierarchical memory operates with >1000 keypoints per frame at 80fps VGA
- Numerically stable 32 bit fixed-point implementation with hardware sharing of different kernels

We thank TSMC University Shuttle Program for chip fabrication and Samsung for funding support

Q & A



System demo in DS1!!!

A 2.1TFLOPS/W Mobile Deep RL Accelerator with Transposable PE Array and Experience Compression

Changhyeon Kim, Sanghoon Kang,
Dongjoo Shin, Sungpill Choi, Youngwoo Kim,
and Hoi-Jun Yoo

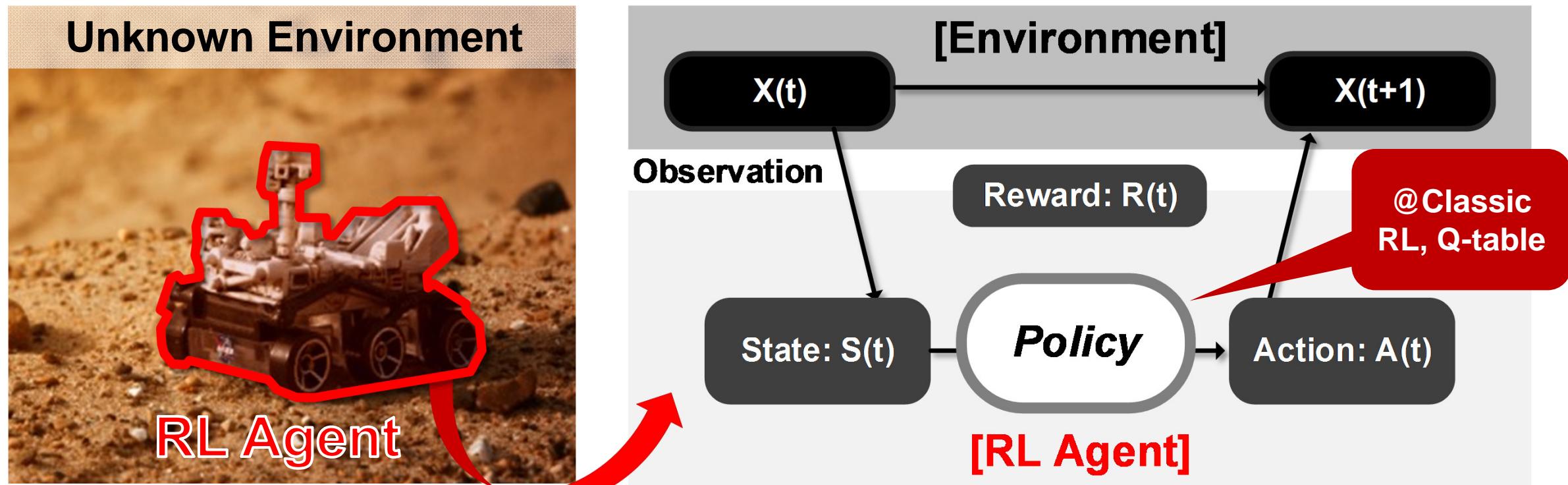
School of Electrical Engineering, KAIST

Outline

- I. Motivation – Deep Reinforcement Learning (Deep RL)
- II. Overall Architecture
- III. Key Features
 - I. Experience Compressor
 - II. Adaptive Data Reuse Transposable PE Array
- IV. Evaluation Results
- V. Conclusion

Reinforcement Learning (RL)

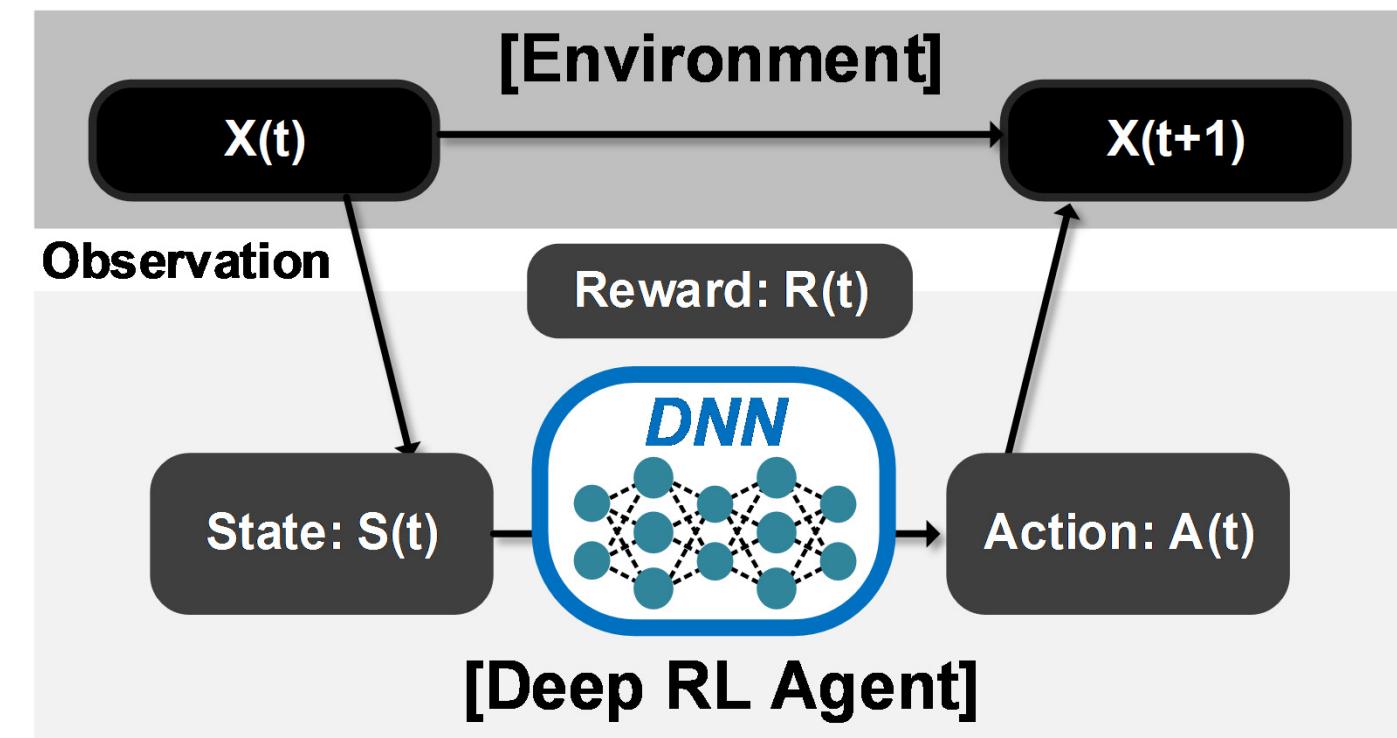
- Train RL Agent @ Unknown Environment: '*Trial-and-Error*'
 - ‘Policy’: Agent strategy to determine ‘Action’ based on ‘State’
 - Goal: Find the optimal ‘Policy’ to maximize ‘Reward’
 - In Classic RL, hard to find optimal policy for complex tasks



Deep Reinforcement Learning (Deep RL)

□ Deep RL: Deep Learning + Reinforcement Learning

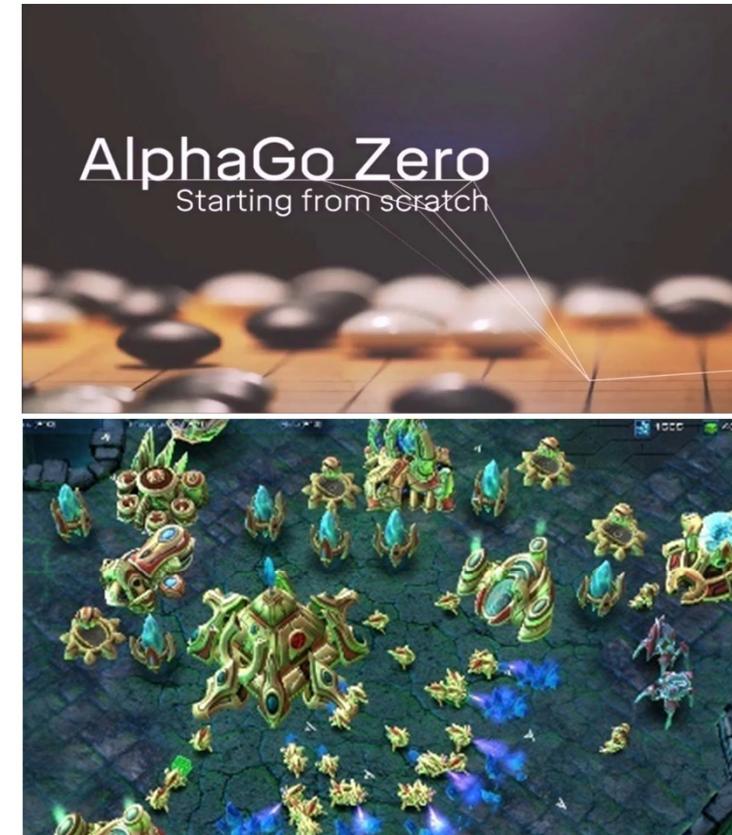
- Using '**Deep Neural Network**' as '**Policy**'
- Deep Learning with no user-supervised labels
→ Find Optimal '**Action**' for **complex tasks**



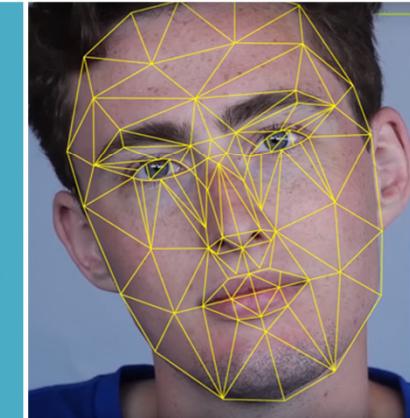
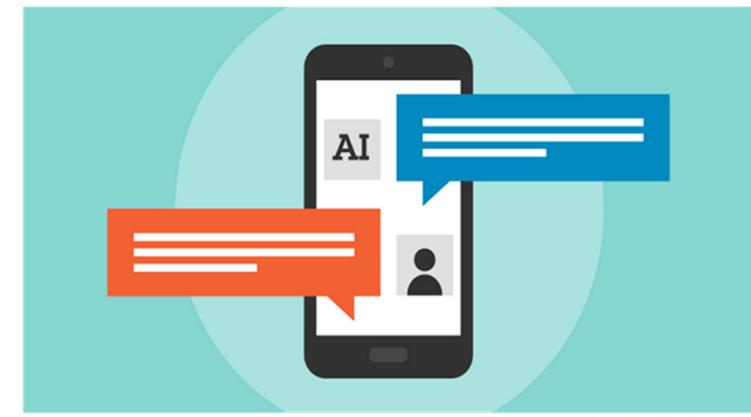
Deep RL Real World Applications



**Continuous
Control Tasks**



Gaming A.I.



**Chat-bot
Q&A**

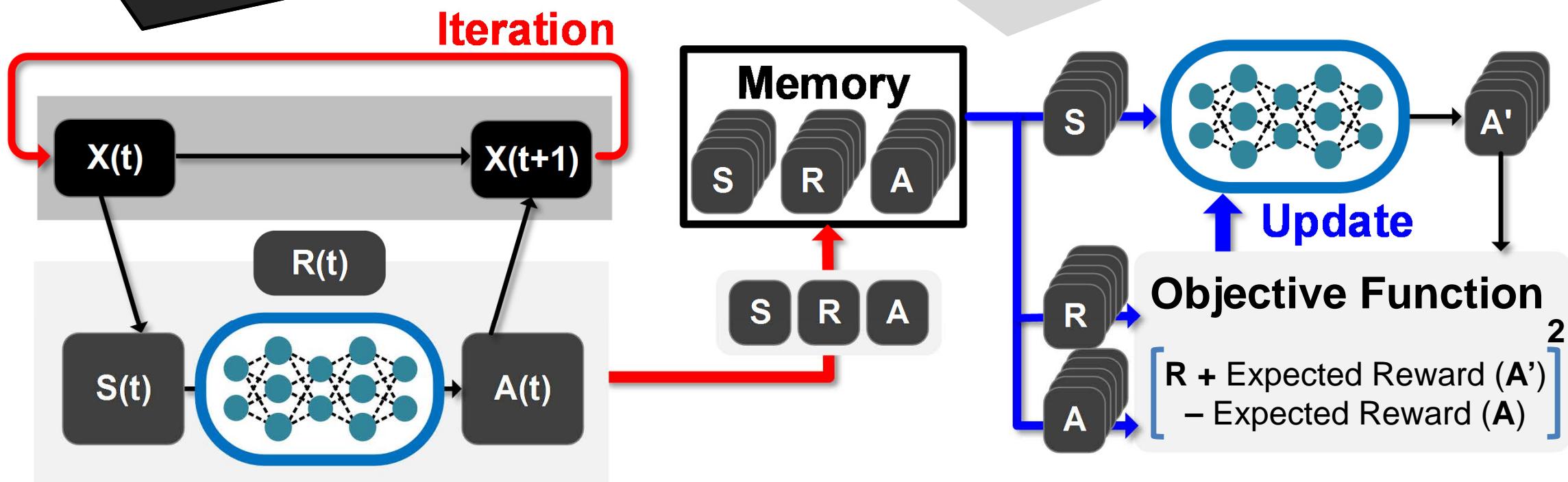
Deep RL Processing Flow

Step 1 – Sample Collection (SC)

DNN Inference to determine ‘Action’
 Store (‘State’, ‘Reward’, ‘Action’) =
 ‘Experience’

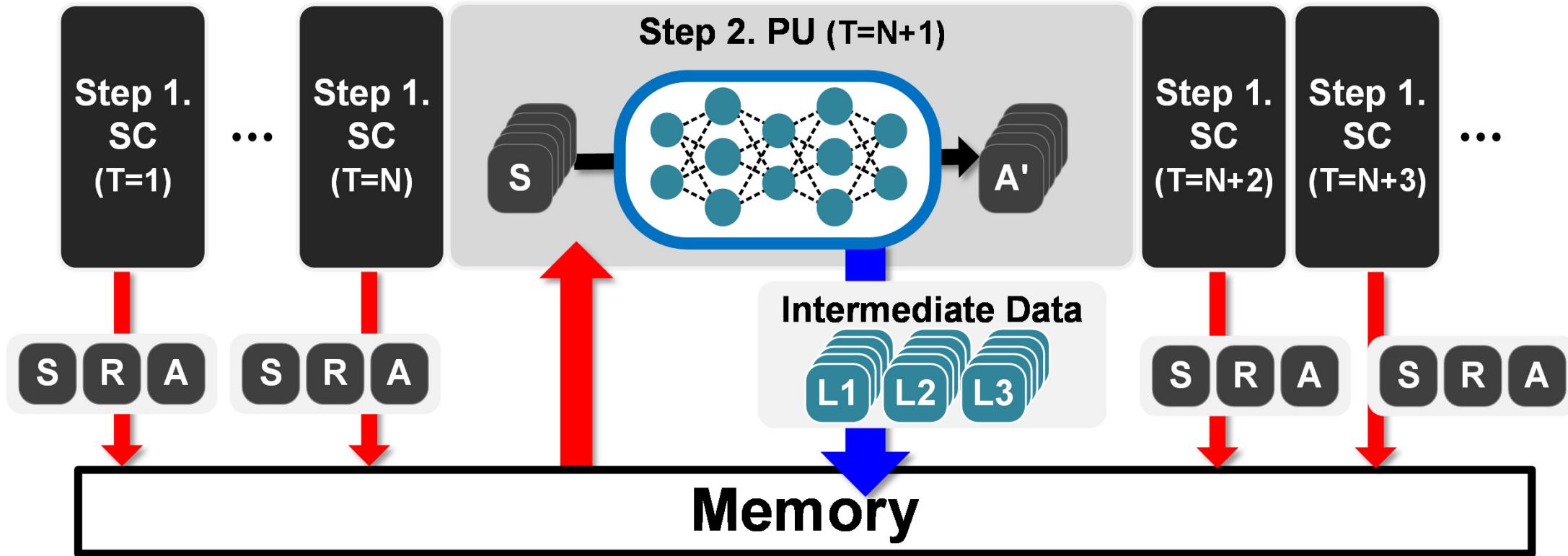
Step 2 – Policy Update (PU)

Update DNN to Maximize ‘Reward’
 w/ Previous ‘Experience’



RL Challenge 1: Large Memory Access

- Critical bottle-neck of DNN H/W
- Extra memory bandwidth required for Deep RL
 - Store & Fetch ~10000s ‘**Experience**’ + **Intermediate Data**

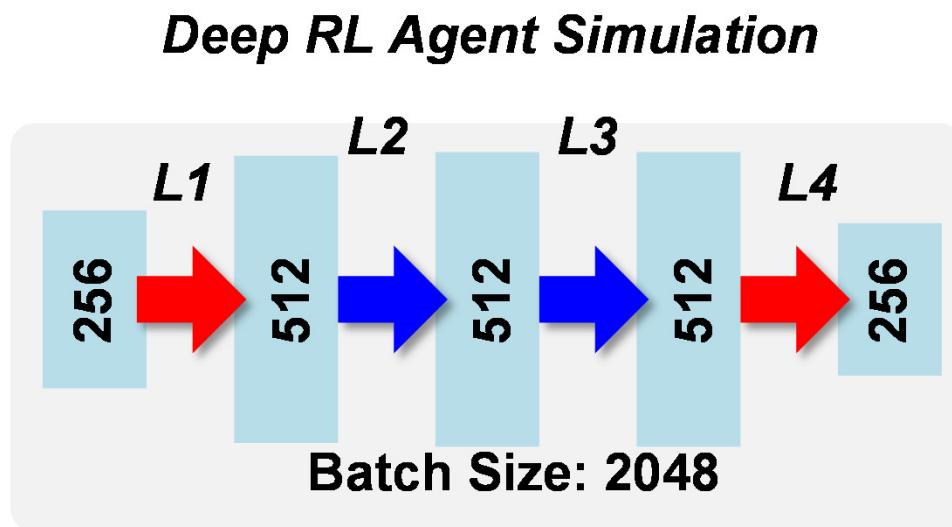


RL Challenge 1: Large Memory Access

☐ Large Memory Access for PU Step

- PU must be processed b/w the previous frame & next frame

: Batch Size↑ → **Stable RL, Large Peak Memory Bandwidth**

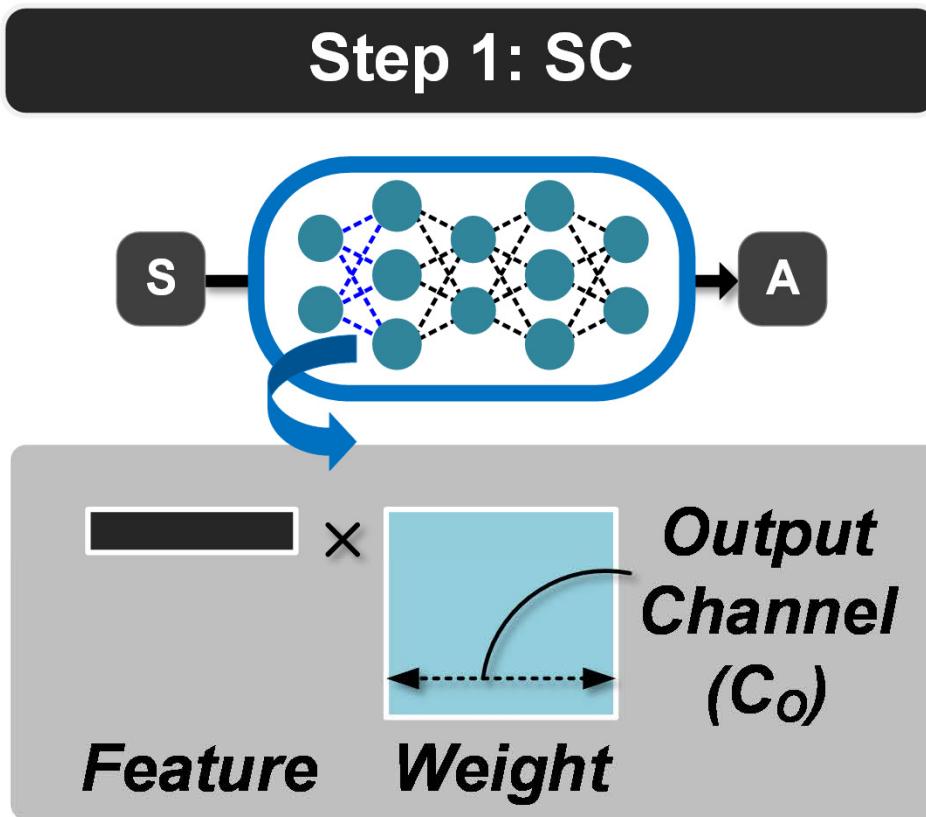


# of Agent	8
FPS	50
Precision	FP16

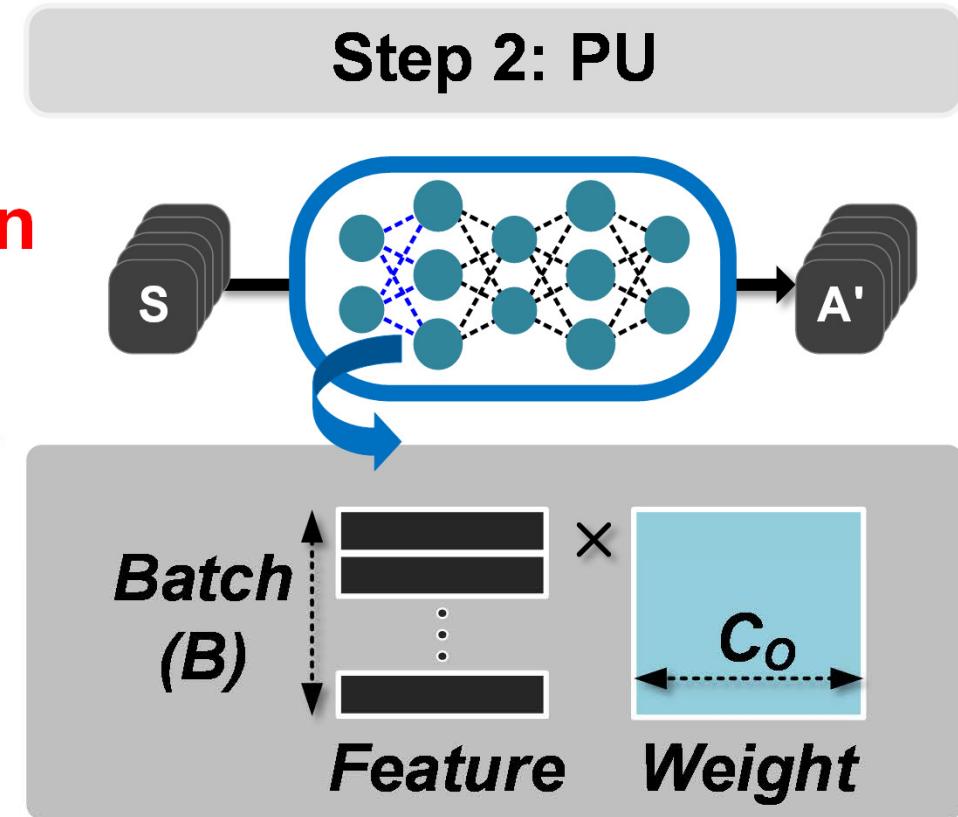


RL Challenge 2: Different Data Pattern

- Different Data Patterns b/w Step 1 & Step 2



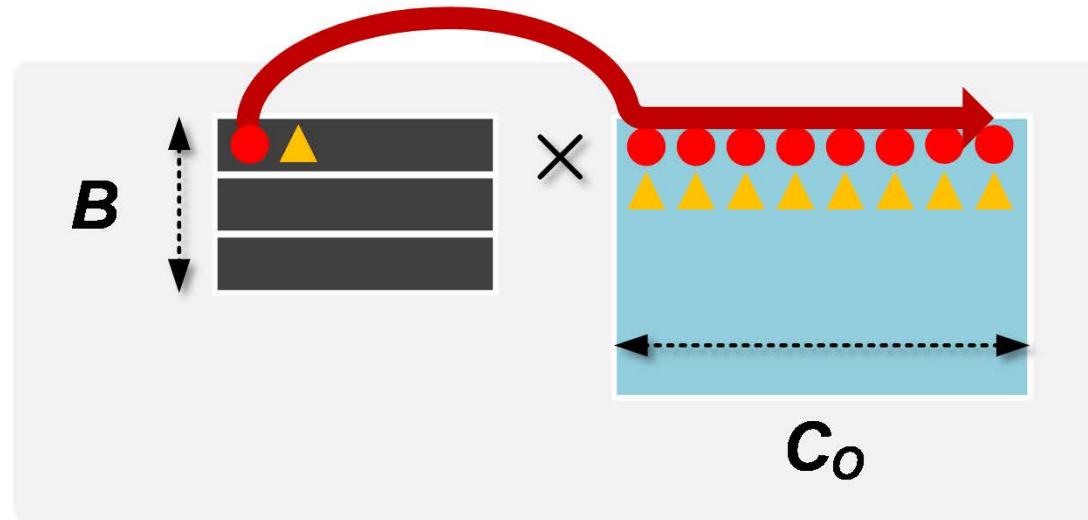
Single Feature \times Weight



Multiple Feature \times Weight

RL Challenge 2: Different Data Pattern

of Feature Reuse: C_o



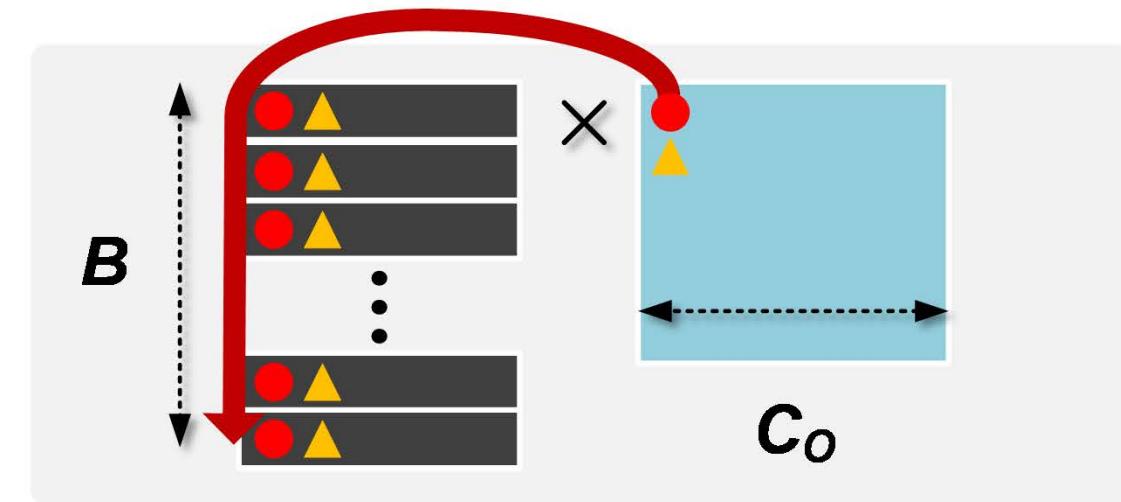
Step 1: SC

In General,

$\text{Small } B < \text{Large } C_o$

Feature Reuse Efficient

of Weight Reuse: B



Step 2: PU

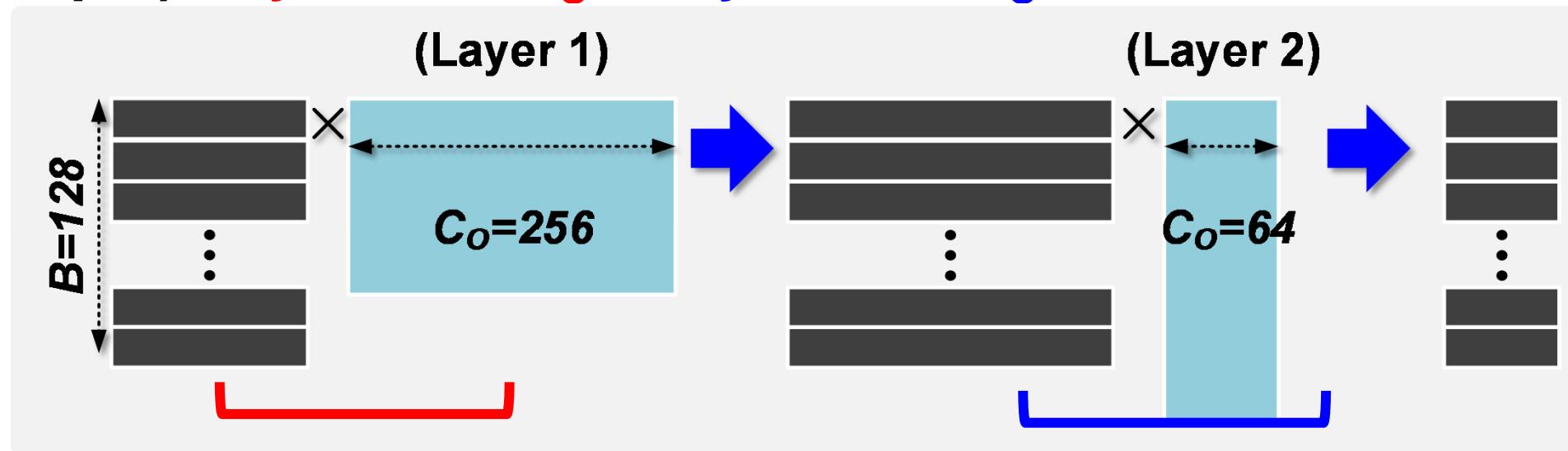
$\text{Large } B > \text{Small } C_o$

Weight Reuse Efficient

RL Challenge 2: Different Data Pattern

□ Case: Layer-by-Layer Pattern Switching

Example) Layer1: B<C_O / Layer2: B>C_O



Feature Reuse Efficient

Weight Reuse Efficient

Require **Adaptive Data Reuse** for Low-Power Deep RL Processing

Proposed Mobile Deep RL Accelerator

For Memory Access Reduction

1. *Experience Compressor*

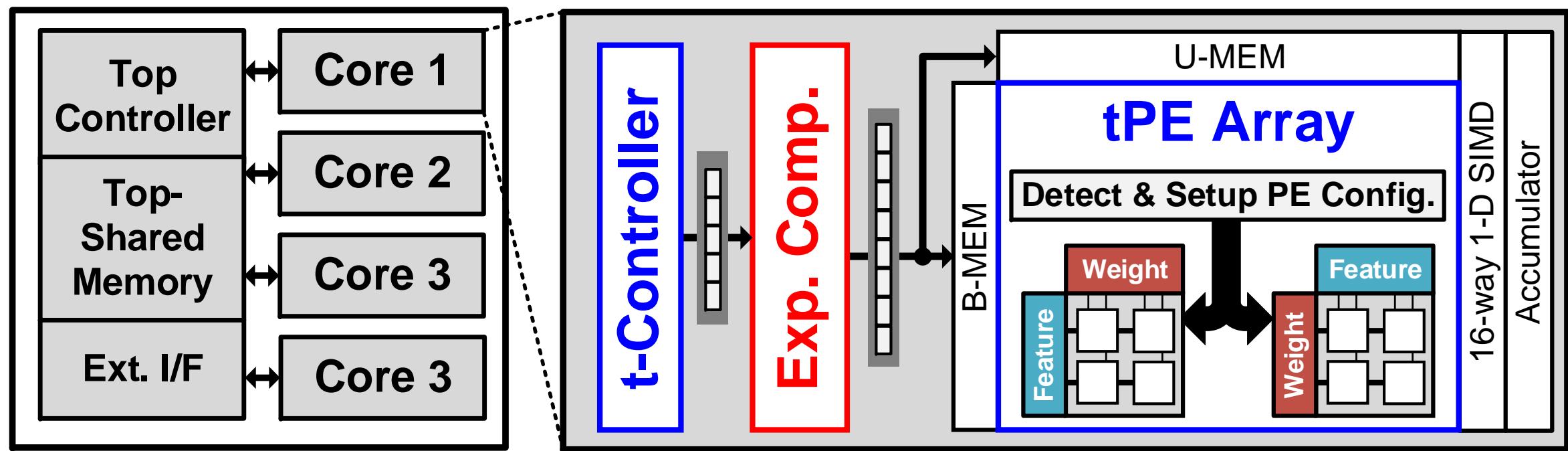
- Store & Fetch Deep RL ‘Experience’ w/ Compressed Format

2. *Transposable PE (tCore) Architecture*

- Adaptive Data Reuse Reconfigurable PE Array

An **2.1TFLOPS/W** Deep RL Processor for **Mobile A.I.**

Overall Architecture



1. Exp. Compressor

- Fetch & Store Compressed Experiences

2. Transposable PE Array

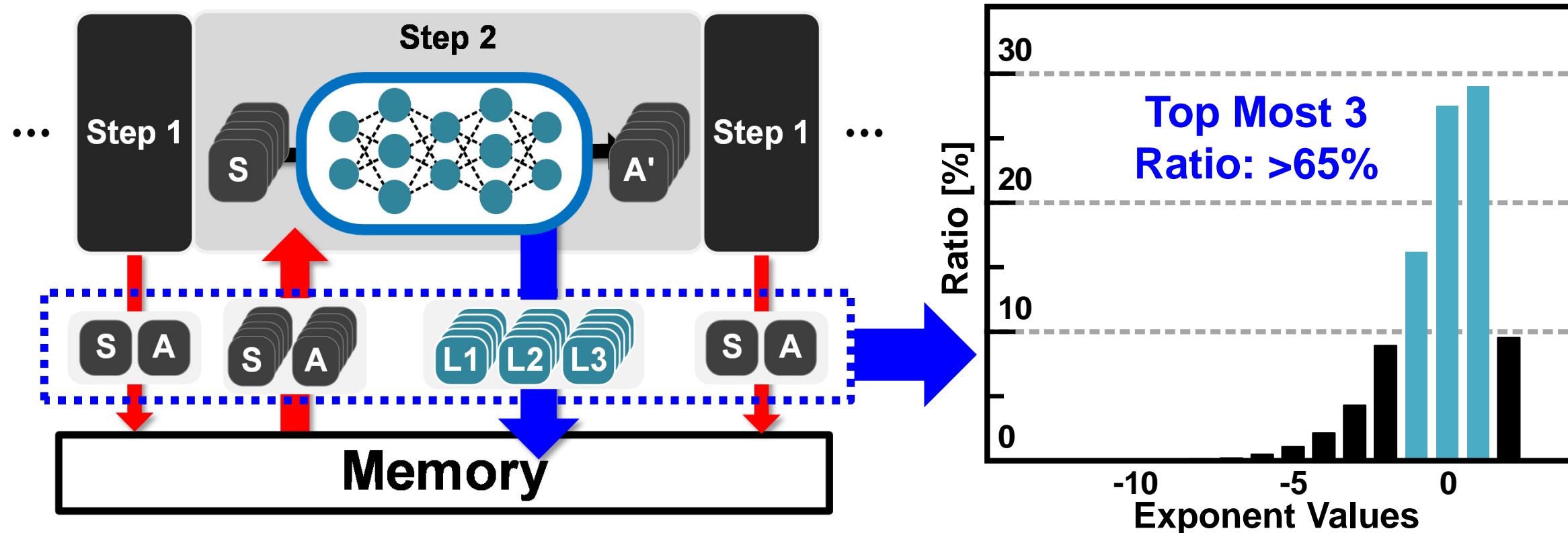
- Parallel DNN Processing w/ Broadcasting & Unicasting Unit

Outline

- I. Motivation – Deep Reinforcement Learning (Deep RL)
- II. Overall Architecture
- III. Key Features
 - I. Experience Compressor
 - II. Adaptive Data Reuse Transposable PE Array
- IV. Evaluation Results
- V. Conclusion

Experience Data Pattern Analysis

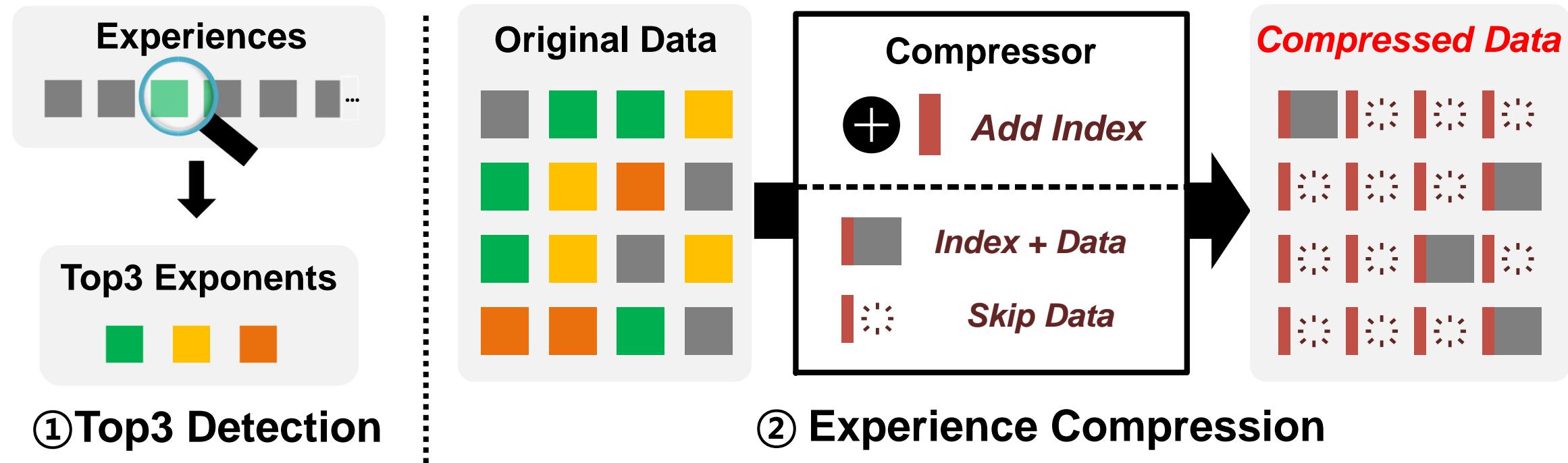
□ Large Memory & Bandwidth Required to Access Experiences



– Experiences: >65% Exponent Values Occupy Top Most-3

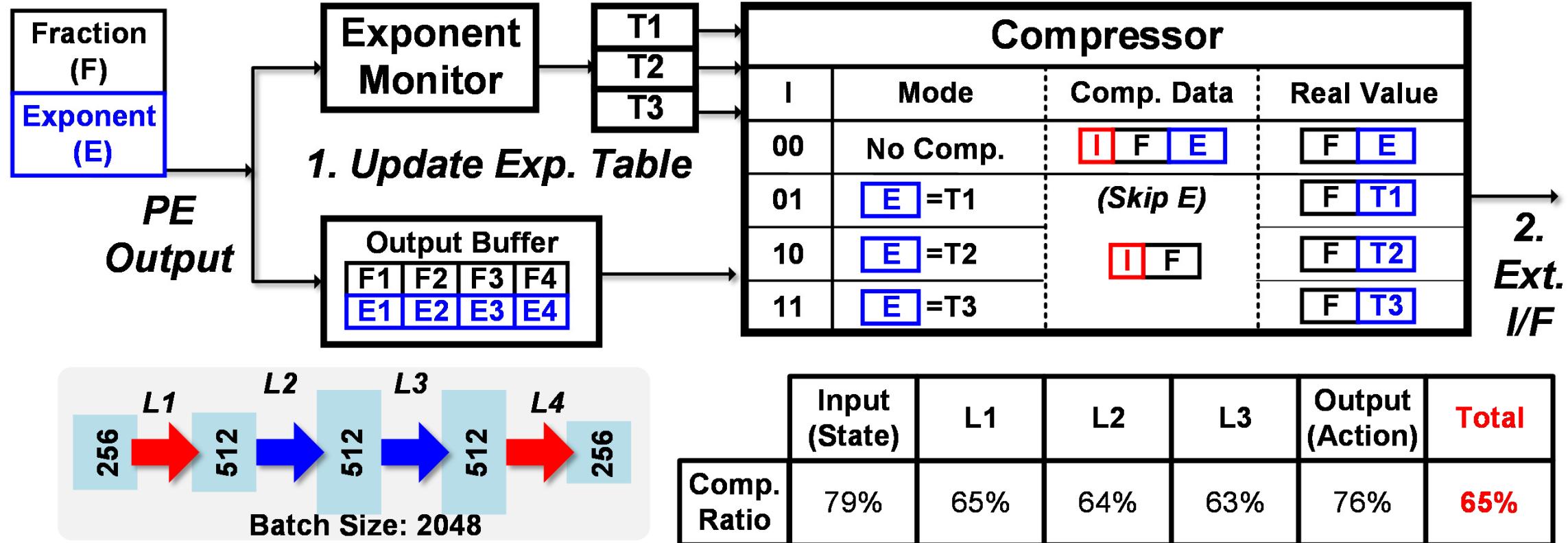
Experience Compression Method

❑ Extra Index to Skip Exponents: Indicate Exponent Value



- **On-line (On-chip) Top-3 Selection** during Sample Collection
 - Memory Bandwidth **Dynamically Reduced** to Current Input

Experience Compressor



□ Layer-by-Layer On-chip Compression

- Exponent Table (T1, T2, T3) Updated, While Output Fetched
- Ext. I/F: Store & Fetch Only Compressed Features → **65% of Original Data**

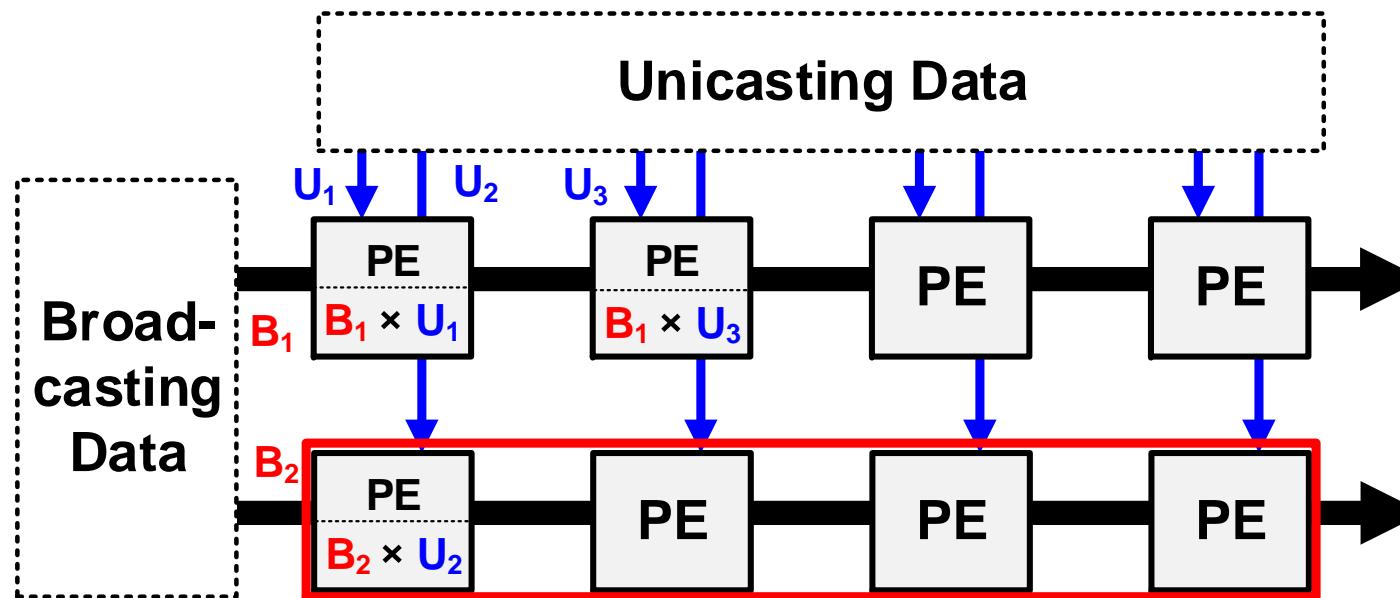
Outline

- I. Motivation – Deep Reinforcement Learning (Deep RL)
- II. Overall Architecture
- III. Key Features
 - I. Experience Compressor
 - II. Adaptive Data Reuse Transposable PE Array
- IV. Evaluation Results
- V. Conclusion

Data Broadcasting & Unicasting

□ 2D-Mesh type PE Array w/ **Broadcasting** & **Unicasting**

Each PE Stores Different Unicasted Data



Fetch New Shared Data Every-cycle

← → **Array Width (PE_w)**

Broadcasting Data:

Fetch New Data Every Single Cycle

of Data Reuse: **Array Width** (PE_w)

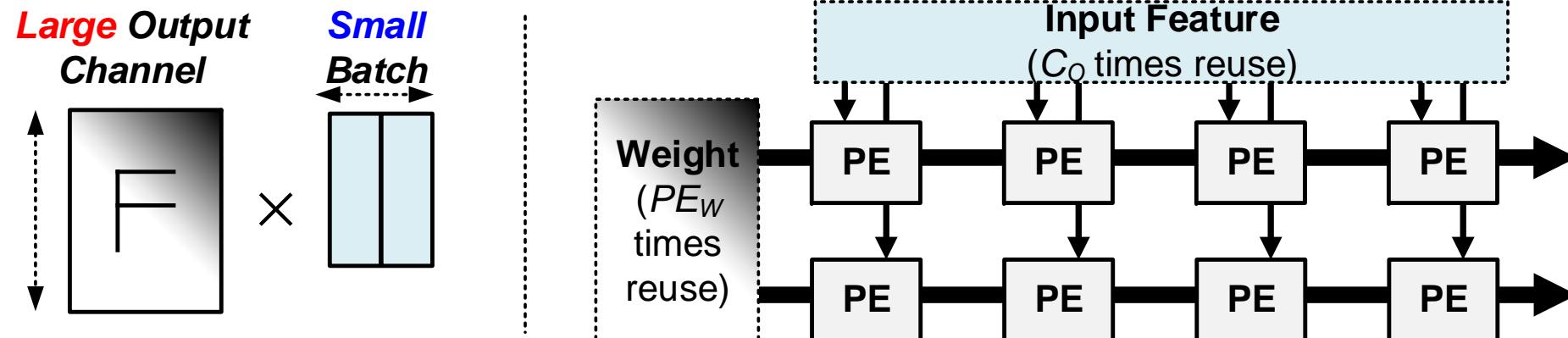
Unicasting Data:

Stored in PE until Fully-reused

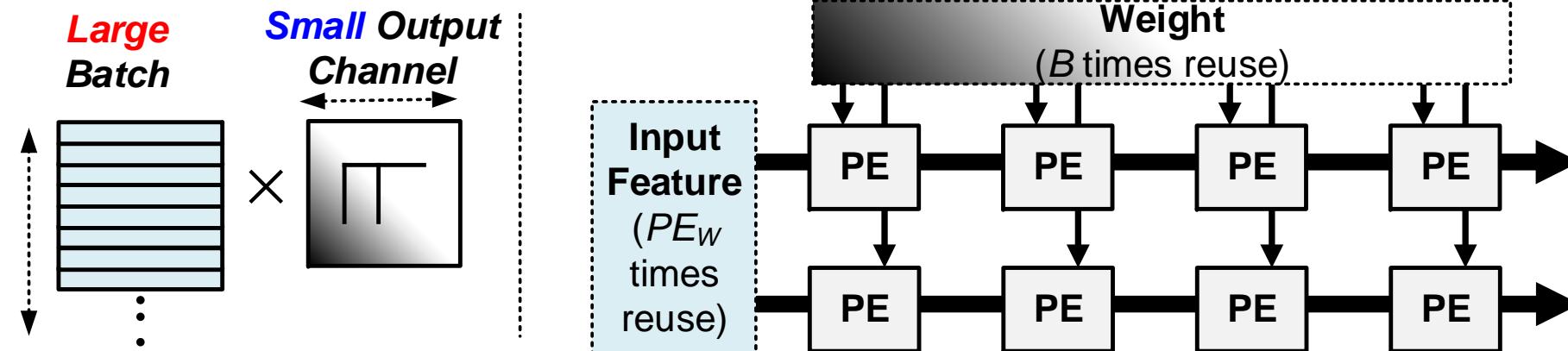
of Data Reuse: **Output Channel** or
Batch

Transposable PE Architecture

□ Weight Broadcasting (WBC) – Feature Reuse

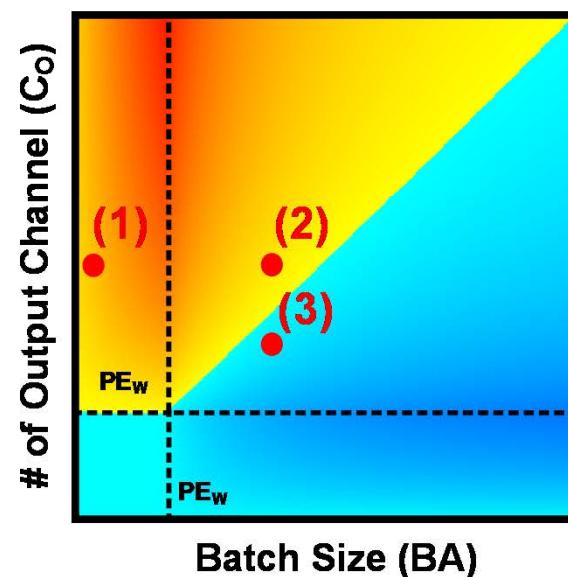


□ Feature Broadcasting (FBC) – Weight Reuse

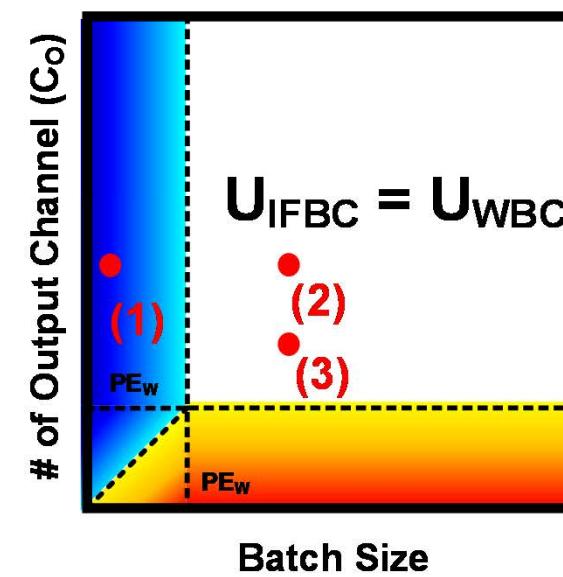


Memory Access & PE Utilization

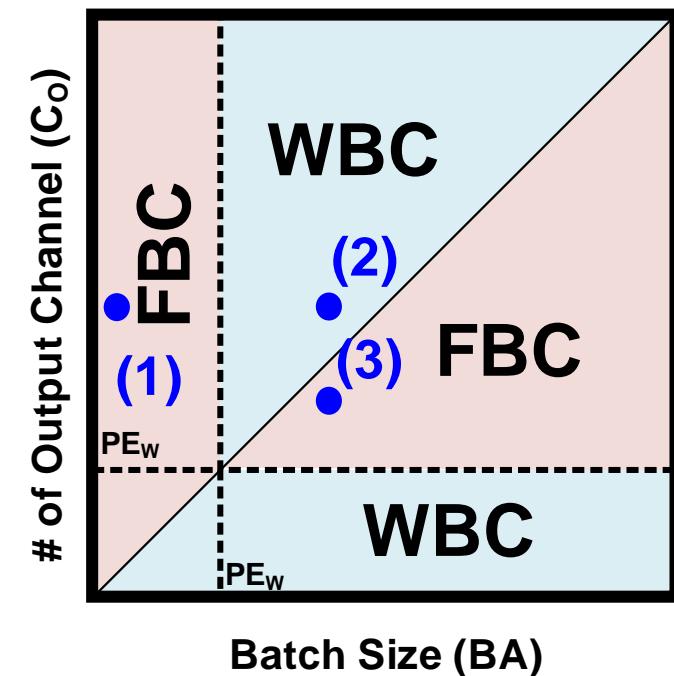
(Memory Access)



(PE Utilization)



(PE Configuration)



Step 1. Sample Collection

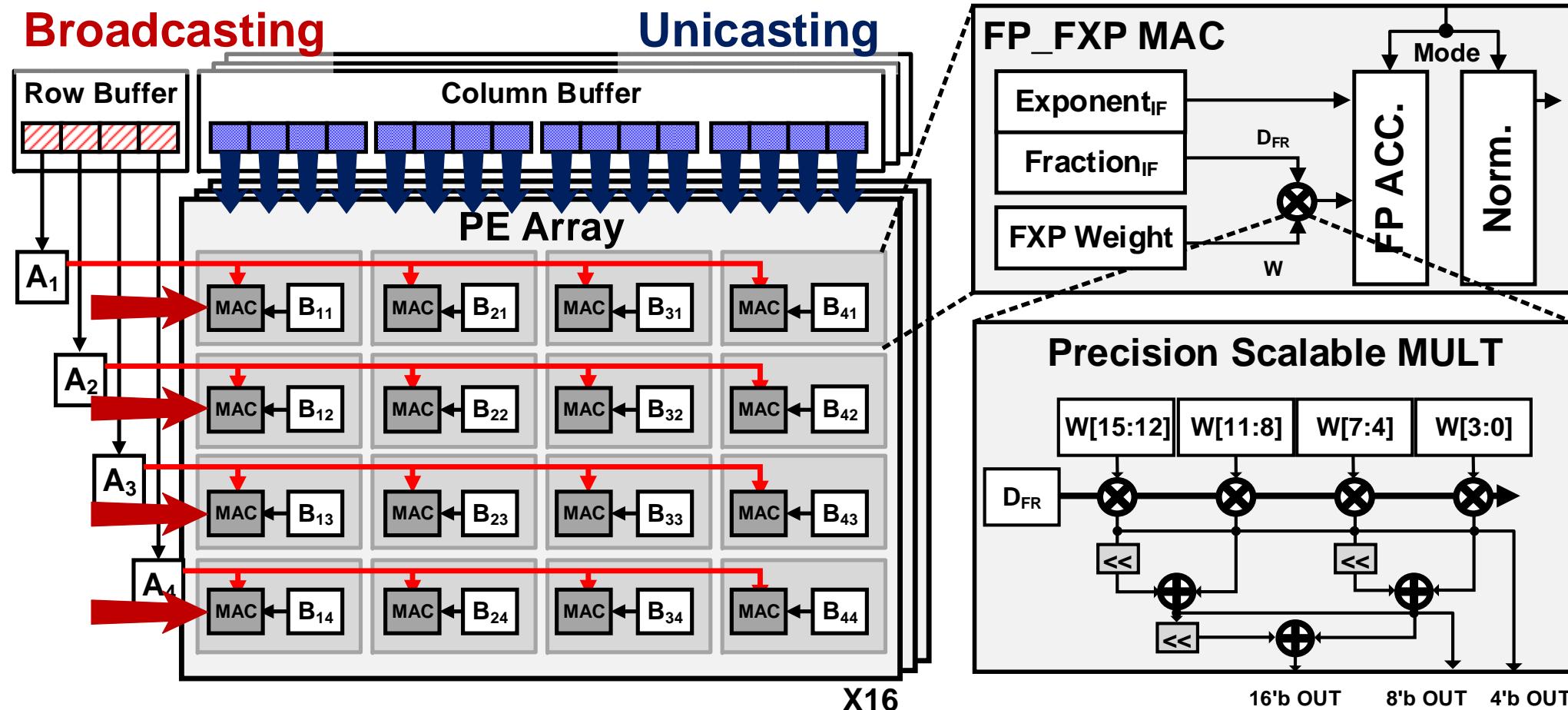
(1) Small Batch \times Multiple Output Channel

Step. 2

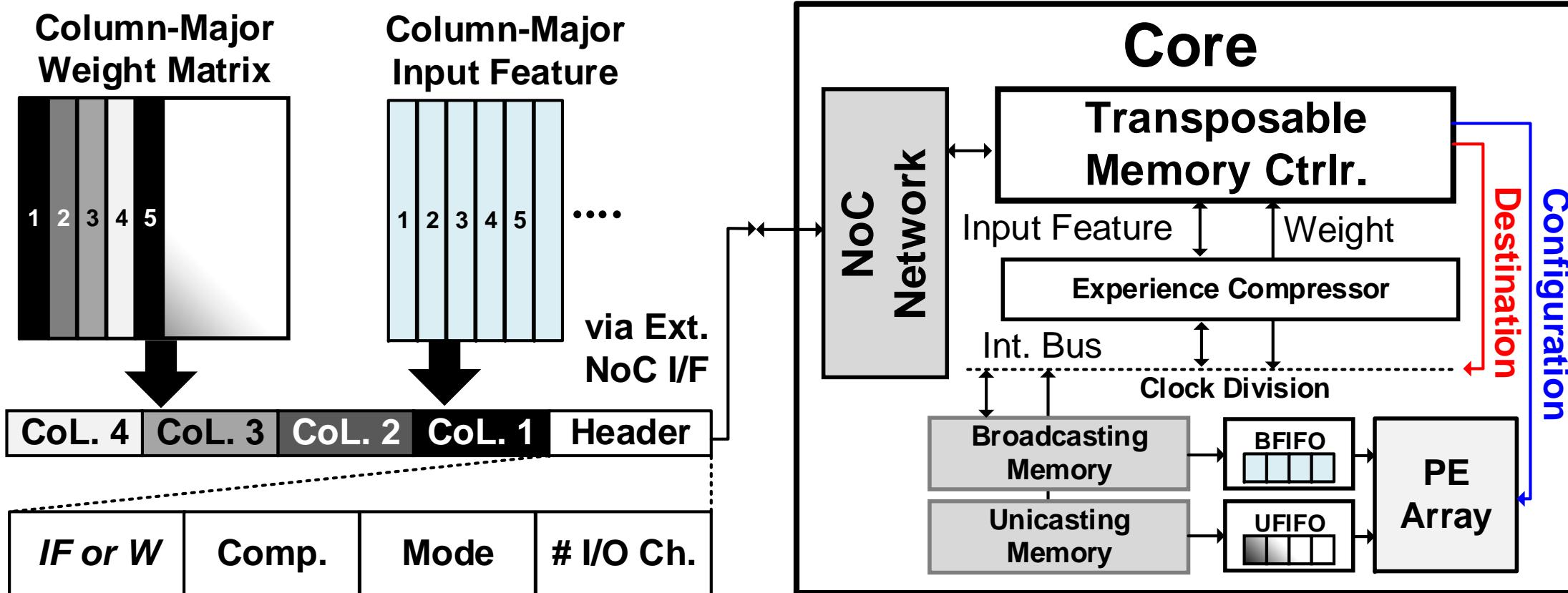
- (2) Batch Size $<$ Output Channel Size
- (3) Batch Size $>$ Output Channel Size

Detailed PE Architecture

□ 16 Parallel PE Array w/ 4x4 Precision Scalable PE



Transposable Memory Ctrlr: tPE Reconfiguration



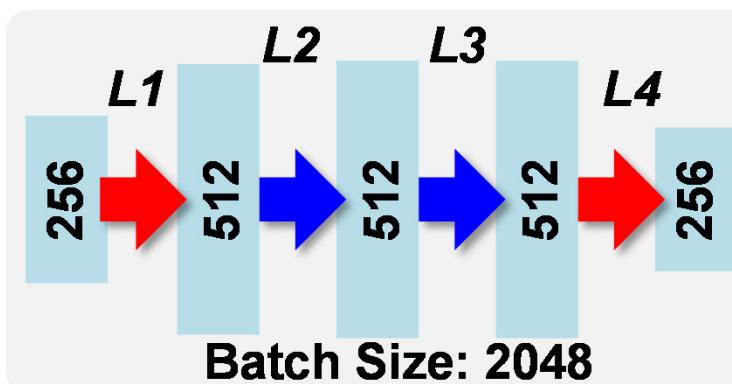
Transposed Memory Controller: Setup PE Configuration & Data Destination
Configure Data Reuse Pattern WBC or FBC w/o User Supervision

tPE Array Power & Bandwidth Reduction

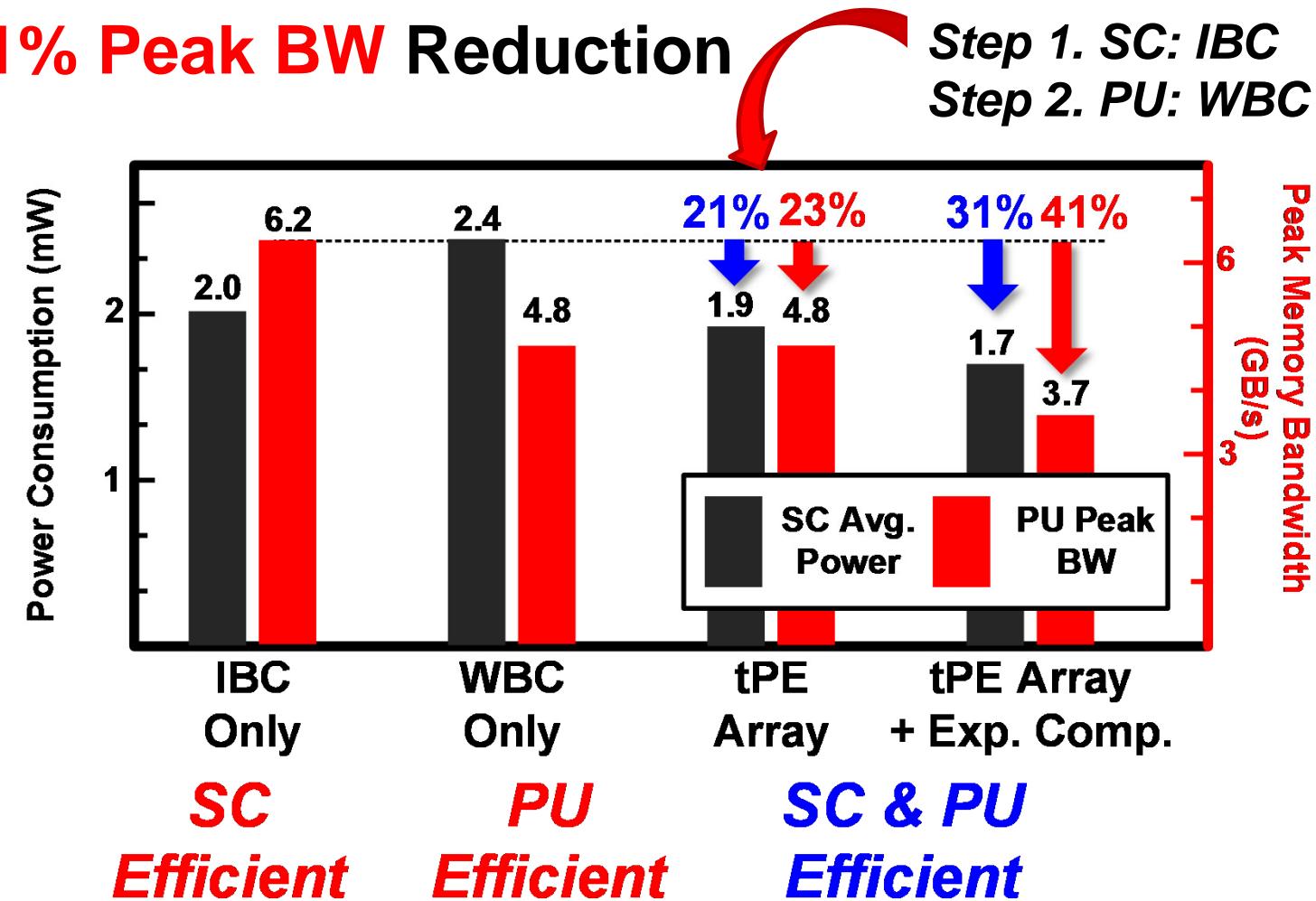
□ Total 31% Power & 41% Peak BW Reduction

Step 1. SC: IBC
Step 2. PU: WBC

Deep RL Agent Measurement



# of Agent	8
FPS	50
Precision	Feature: bfloat16 Weight: int16

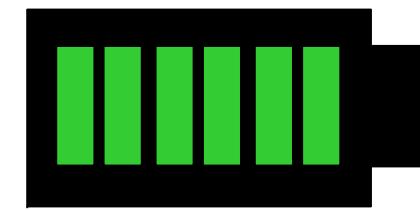
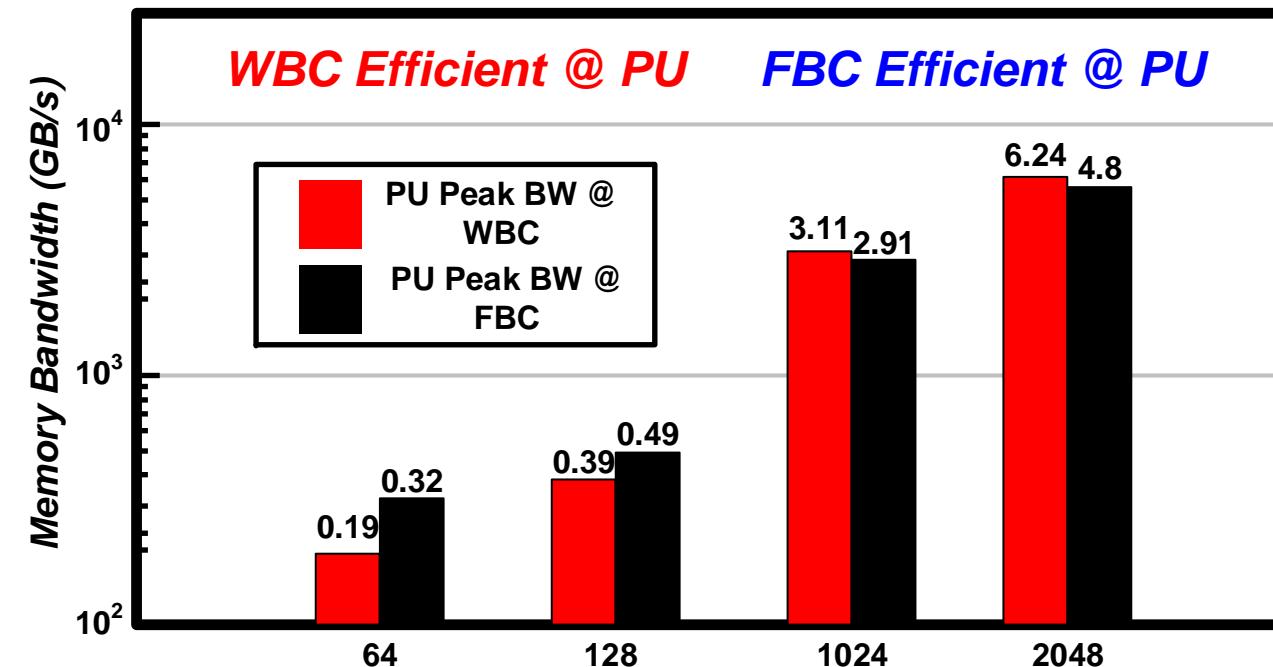


Low-Power & High-Performance Mode

□ Voltage/Frequency/Batch Scaling with Battery Condition



Low Frequency
Low Memory BW
DRL w/ Small Batch



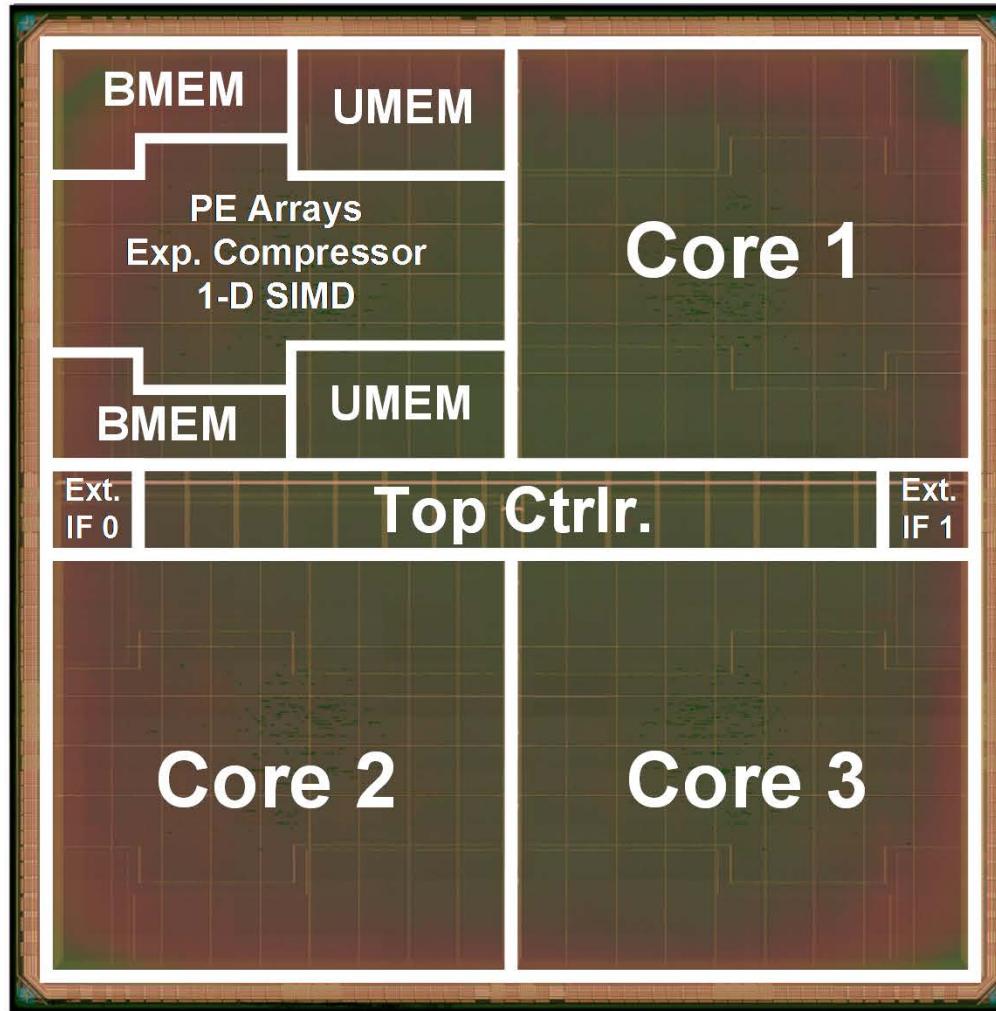
High Frequency
High Memory BW
DRL w/ Large Batch

Quick Response



Highly-Accurate

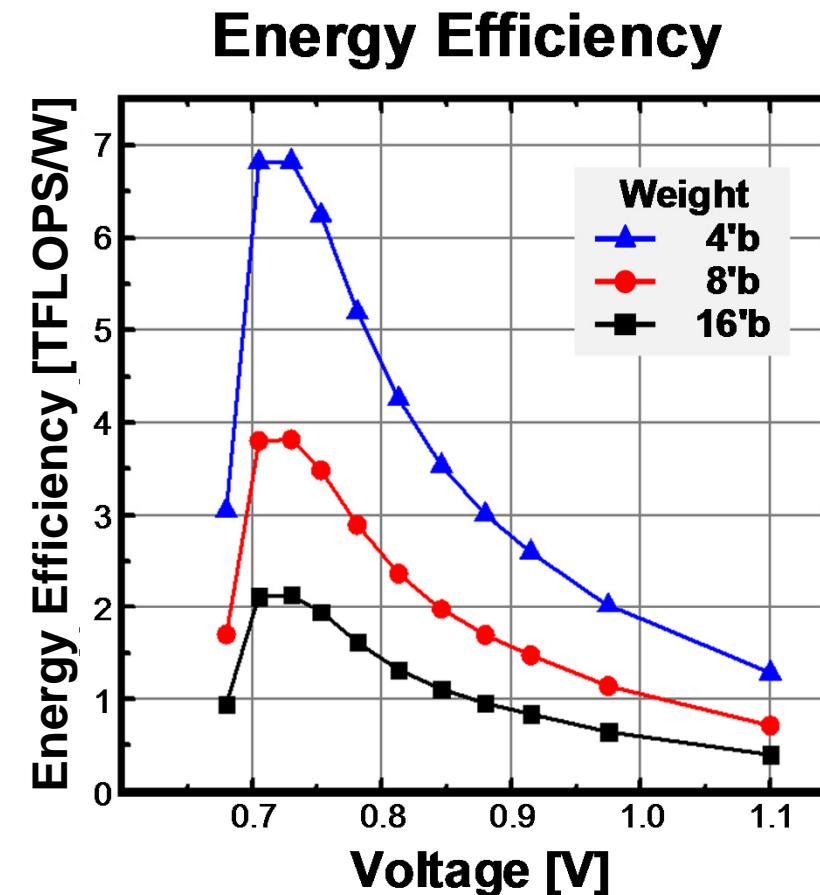
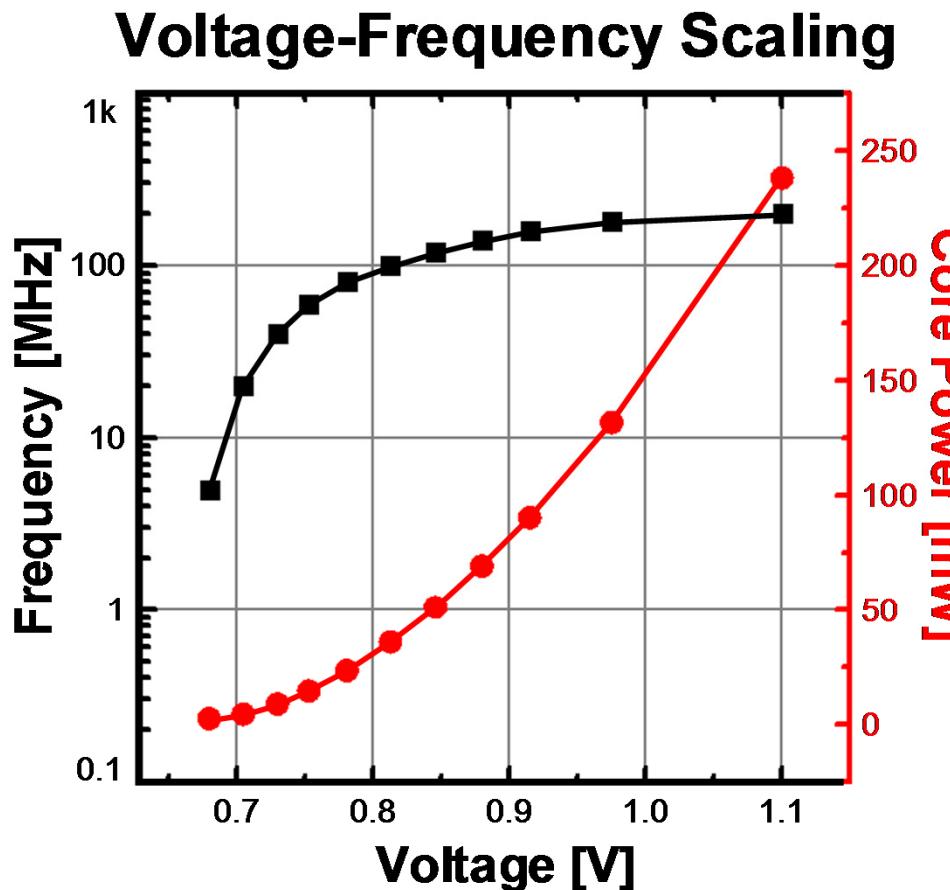
Chip Summary



Process	65nm 1P8M Logic CMOS
Area	4mm × 4mm
SRAM	448 KB
Supply	0.67V – 1.1V
Power	196 mW @ 200MHz, 1.1V
	2.4 mW @ 10MHz, 0.67V
Precision	Feature – bfloat16
	Weight – 16/8/4'b FXP
Peak Performance	204 GFLOPS @ 16b Weight

Measurement Results

□ 16'b Weight Peak Energy Efficiency: 2.1TFLOPS/W @ 40MHz



Demonstration Video



Mobile Deep RL
Demonstration

Comparison Table

	ISSCC 2018 [1]	ISSCC 2018 [2]	S. VLSI 2018 [3]	S. VLSI 2018 [4]	This Work
Inference	CNN/RNN/FC	CNN/RNN/FC	CNN/FC	CNN/RNN/FC	CNN/RNN/FC
Learning	-	-	FC Tuning	CNN/RNN/FC	CNN/RNN/FC
Precision	int 1-4	Feature: int 16'b Weight: int 1-16'b	int 8'b	fp16, binary, ternary	Feature: bfloat16 Weight: int 4,8,16'b
Process (nm)	40	65	65	14	65
Data Compression Scheme	X	X	X	X	O
Energy Efficiency	-	3.06 TOPS/W (16'b)	0.411-62.1 TOPS/W	-	2.16 TFLOPS/W (16'b)
Peak Performance	1.96 TOPS (4'b) 7.49 TOPS (2'b)	345.6 GOPS (16'b)	**102-5638 GOPS	1500 GFLOPS	204 GFLOPS (16'b)
Normalized Area Efficiency	*6.11 GOPS/mm ² (int 4'b)	21.6 GOPS/mm ² (int 16'b)	8.5 - 469.83 GOPS/mm ² (int 8'b)	*7.73 GFLOPS/mm ²	12.75 GFLOPS/mm ²

*[1] Quest [2] UNPU [3] STICKER [4] Intel

Conclusion

- An Energy Efficient Mobile Deep RL Processor is proposed for Low-Power Mobile A.I.s
- Key Features: **31% Power & 41% Peak Bandwidth Reduction**
 1. Transposable PE Array
 2. Experience Compressor

An **2.1TOPS/W**, Energy Efficient
Reconfigurable Deep RL Processor

A 65nm 0.39-to-140.3TOPS/W 1-to-12b Unified Neural-Network Processor Using Block-Circulant-Enabled Transpose-Domain Acceleration with 8.1× Higher TOPS/mm² and 6T HBST-TRAM-Based 2D Data-Reuse Architecture

Jinshan Yue¹, Ruoyang Liu¹, Wenyu Sun¹, Zhe Yuan¹, Zhibo Wang¹, Yung-Ning Tu², Yi-Ju Chen², Ao Ren³, Yanzhi Wang³, Meng-Fan Chang², Xueqing Li¹, Huazhong Yang¹, Yongpan Liu¹

¹Tsinghua University, Beijing

²National Tsing Hua University, Hsinchu

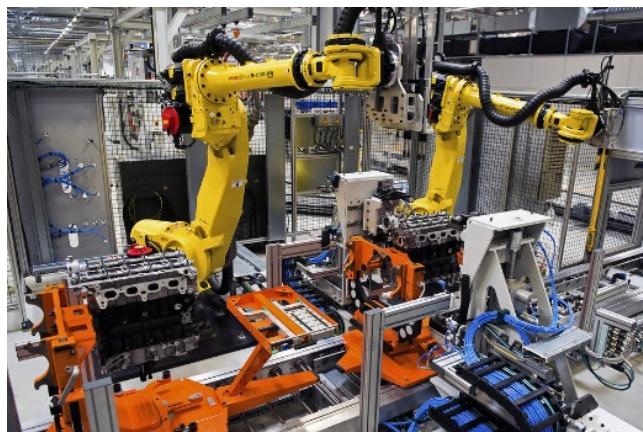
³Northeastern University, Boston, MA



Outline

- ❑ **Introduction**
- ❑ **Challenges of Unified Block-Circulant Processor**
- ❑ **Proposed Unified NN Processor**
 - Global-Parallel Bit-Serial FFT Module
 - 2-D Data-Reuse Array
 - HBST TRAM
- ❑ **Measurement Results**
- ❑ **Conclusion**

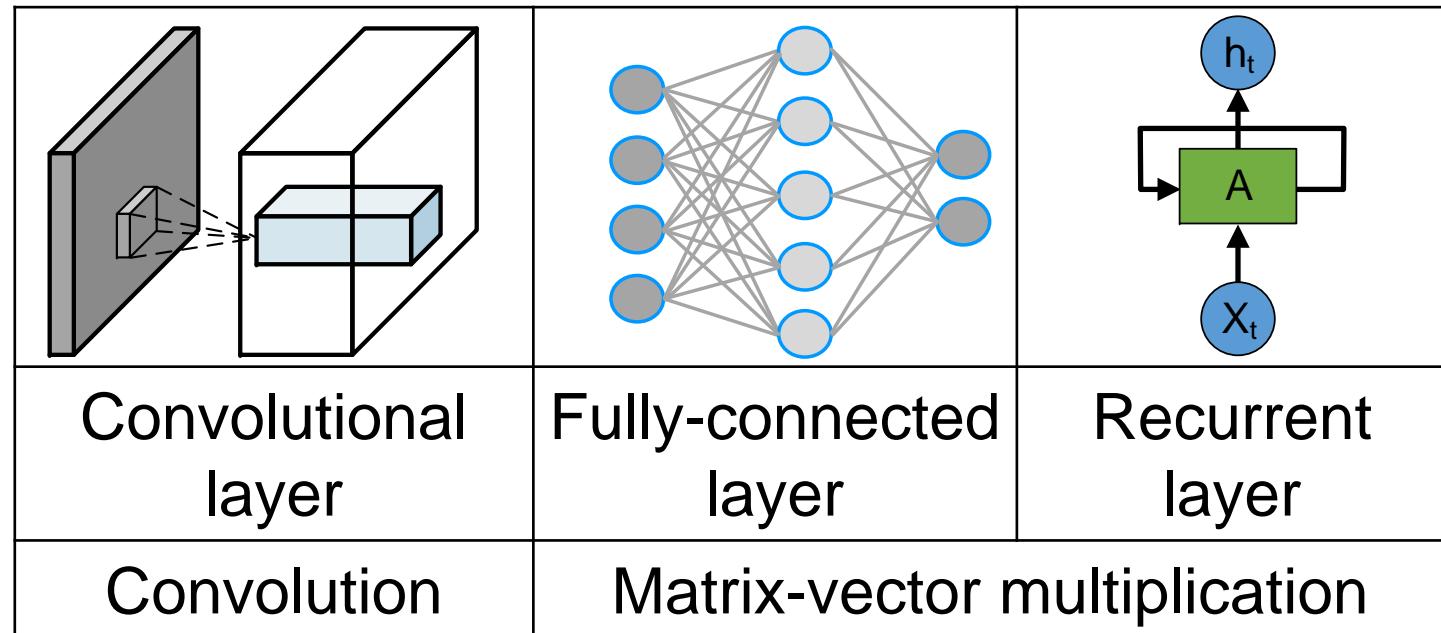
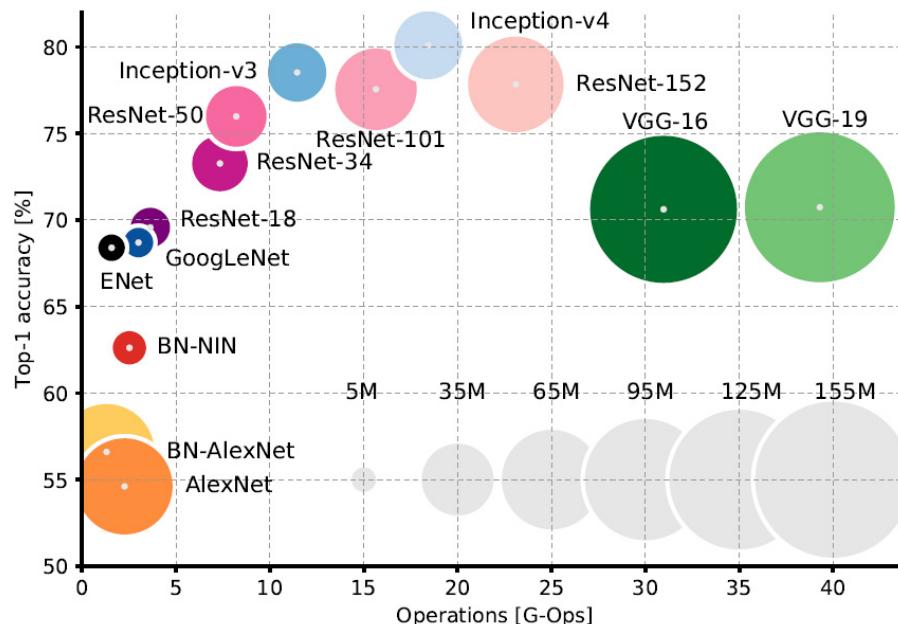
Edge Machine Learning



- Low power
- Low latency
- Low cost

Neural Network Challenges

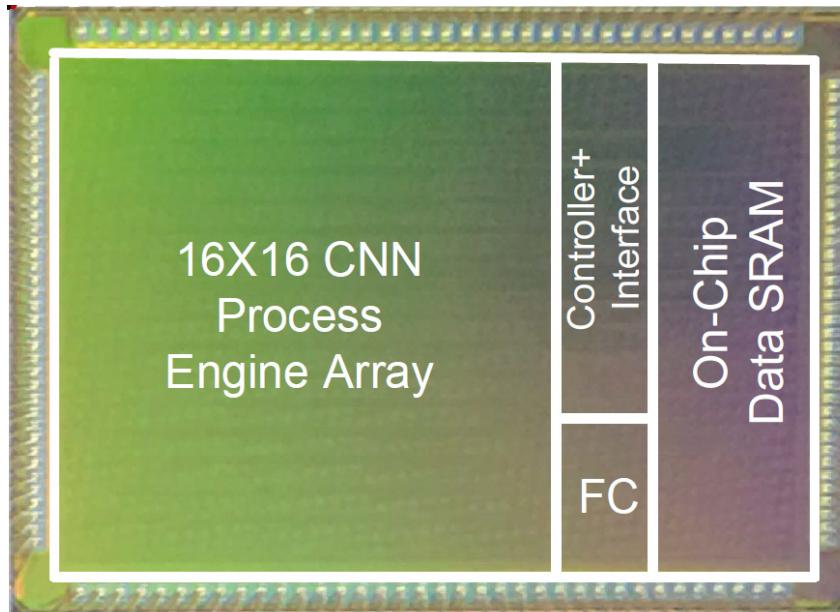
[ArXiv:1605.07678]



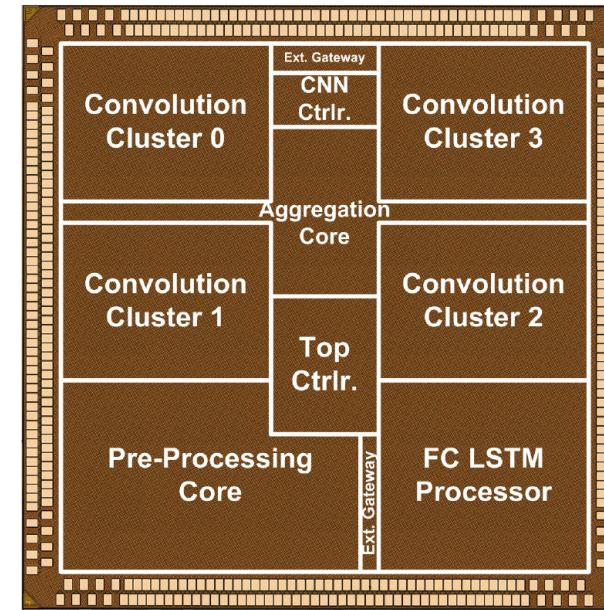
- Computation intensive
- Memory intensive
- Different model types

Efficient unified
neural network processor ?

Heterogeneous NN Processors



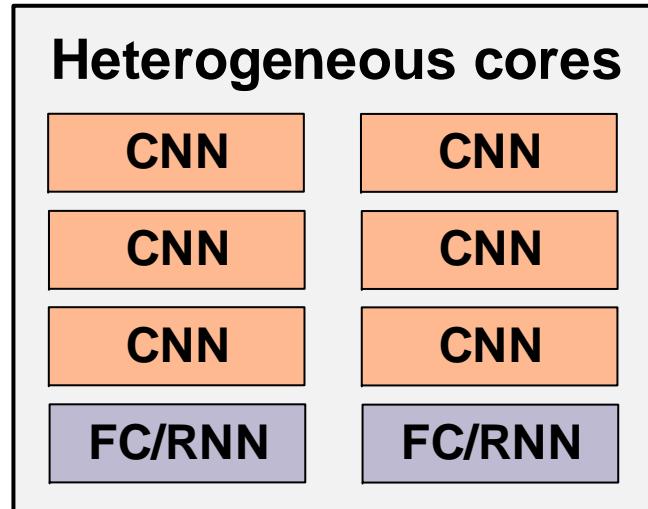
STICKER, Tsinghua, VLSI2018
62.1TOPS/W @CNN core



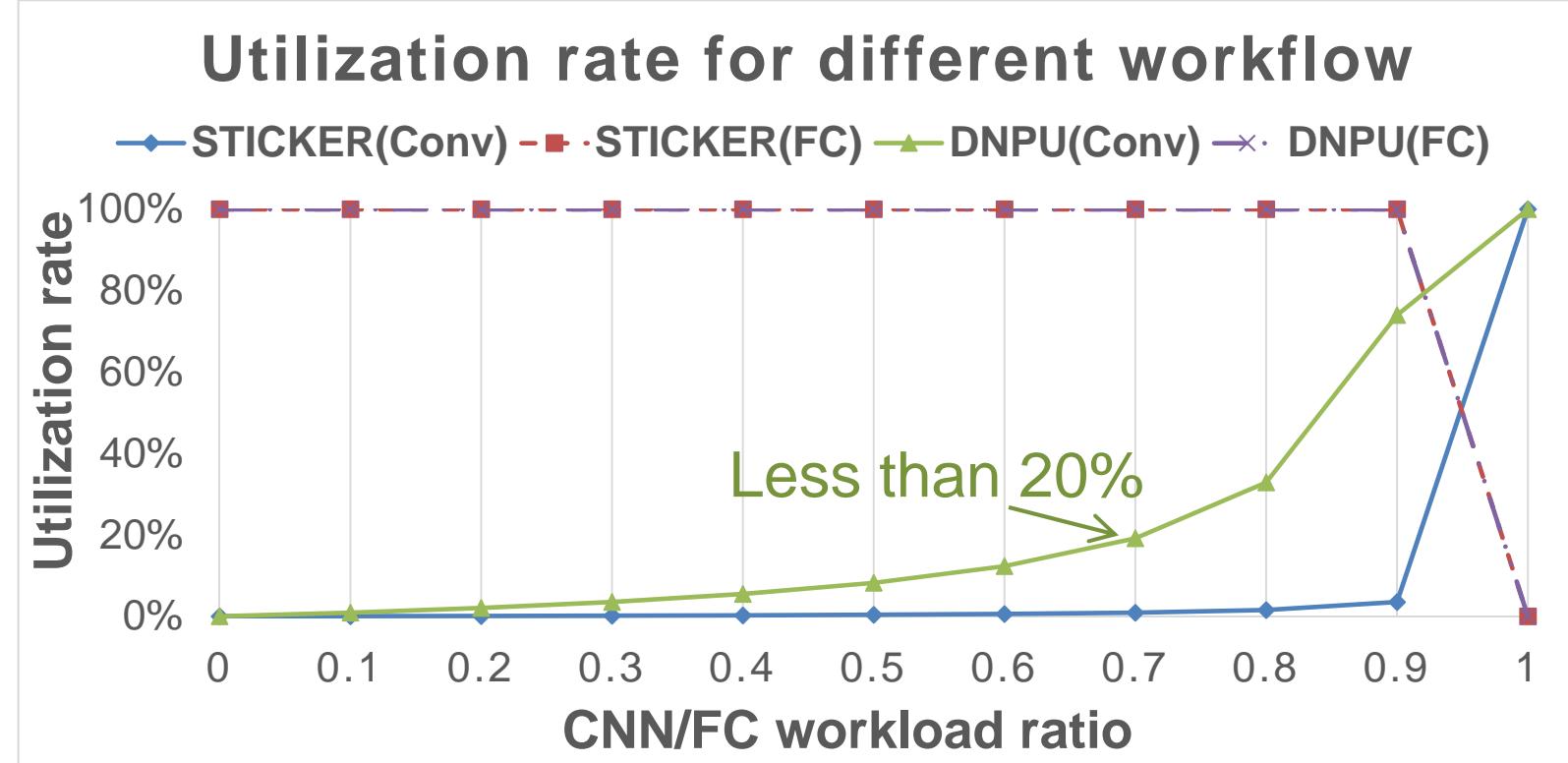
DNPU, KAIST, ISSCC2017
8.1TOPS/W @CNN core

Best energy efficiency on specialized cores

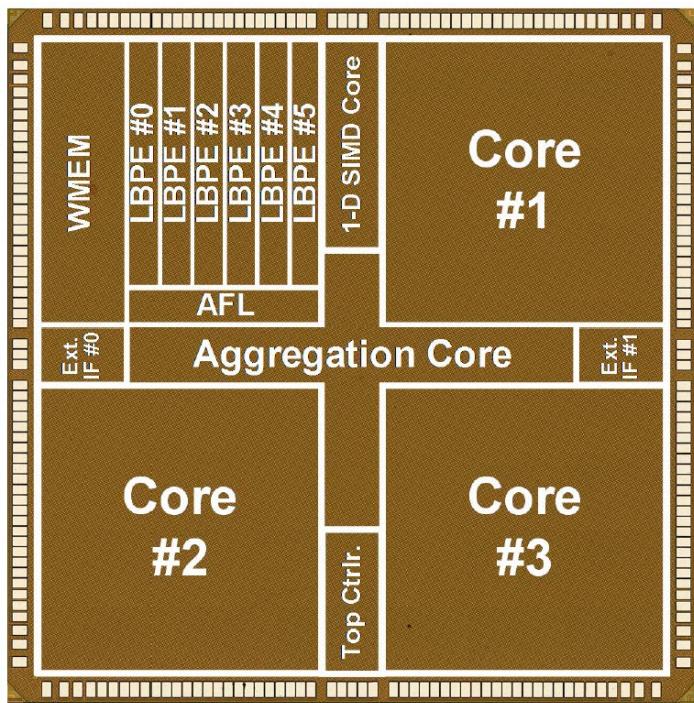
Heterogeneous NN Processors



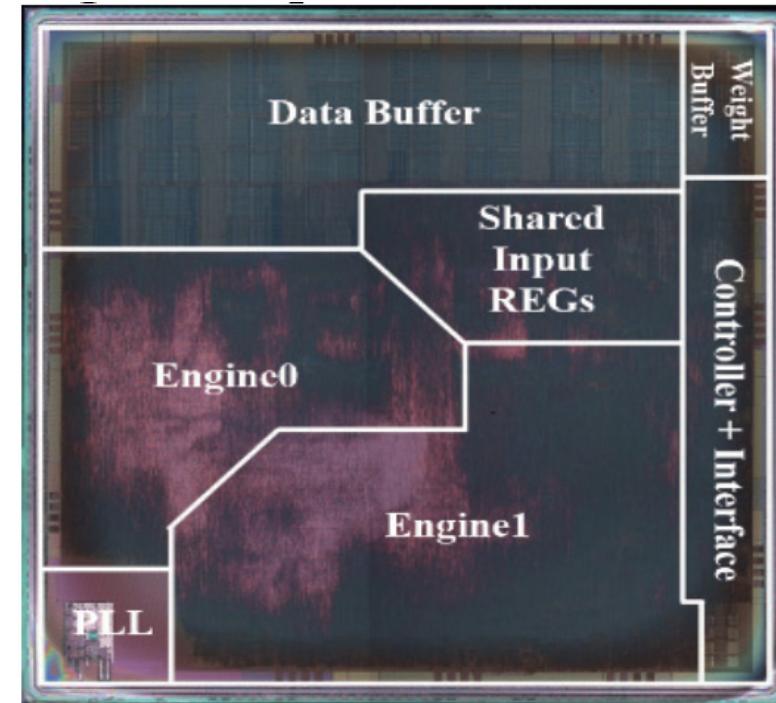
- **Area overhead**
- **Utilization rate**



Reconfigurable NN Processors



UNPU, KAIST, ISSCC2018
11.6TOPS/W @4bit

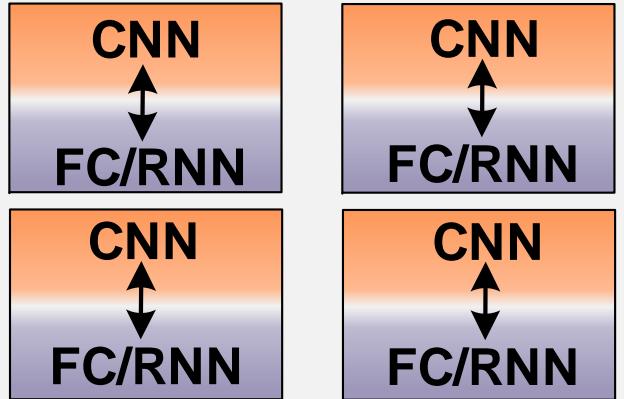


THINKER, Tsinghua, VLSI2017
5.09TOPS/W

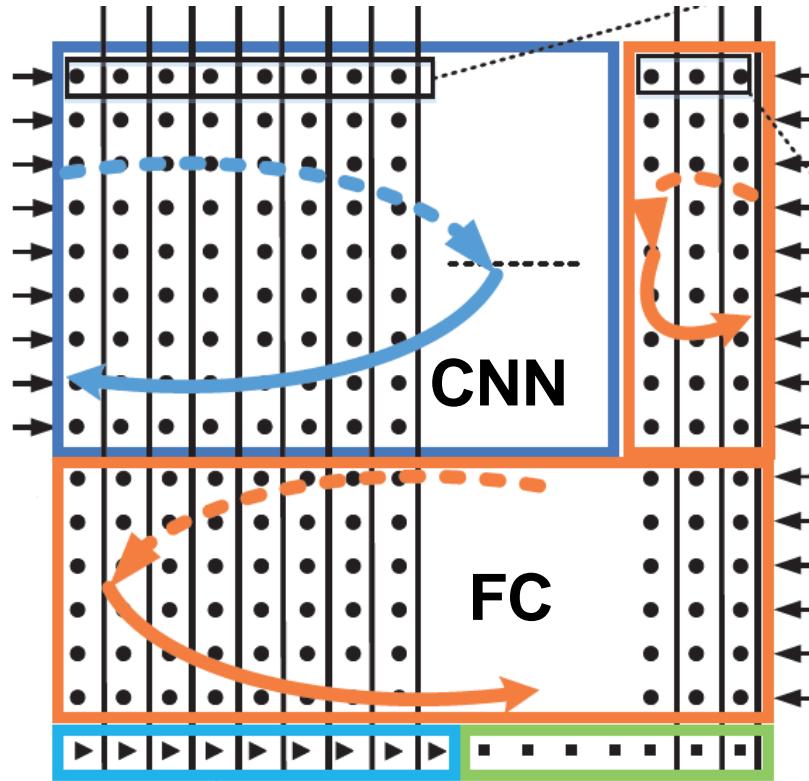
Better hardware utilization due to reconfiguration

Reconfigurable NN Processors

Reconfigurable cores

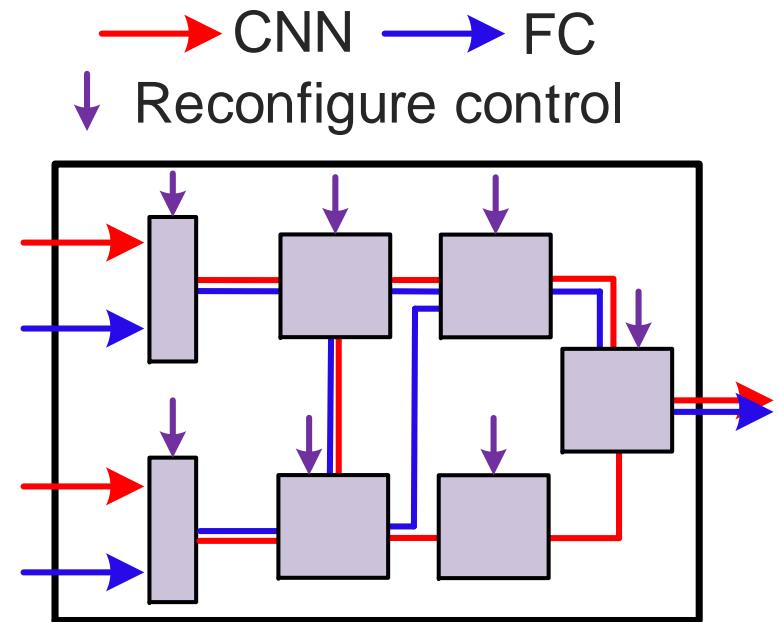


Area overhead (7%)
Limited efficiency
Compiler/scheduling complexity



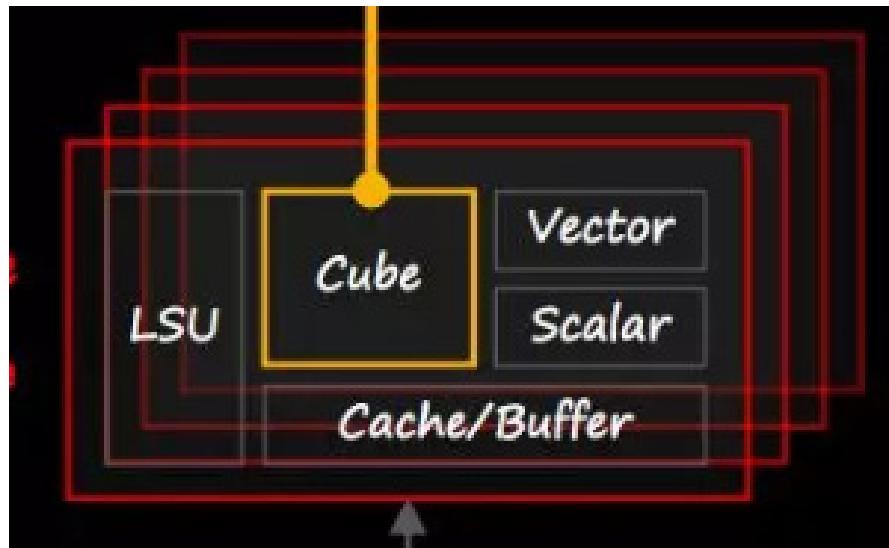
Reconfigurable PE array & PE unit

[VLSI 2017]

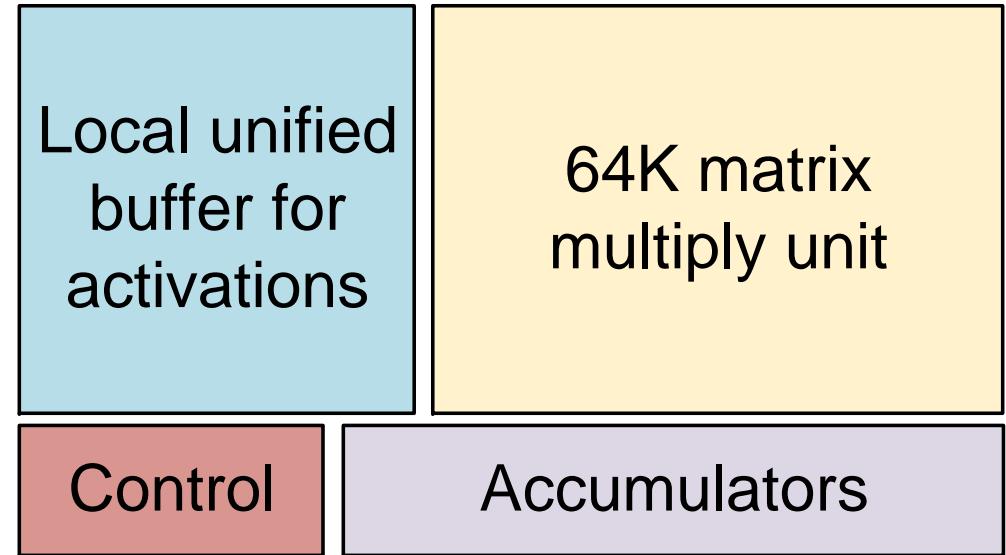


Industrial NN-Specific Processors

Scalable general processing units



Server : Ascend 910 512TOPS@350W
Edge : Ascend 310 16TOPS@8W

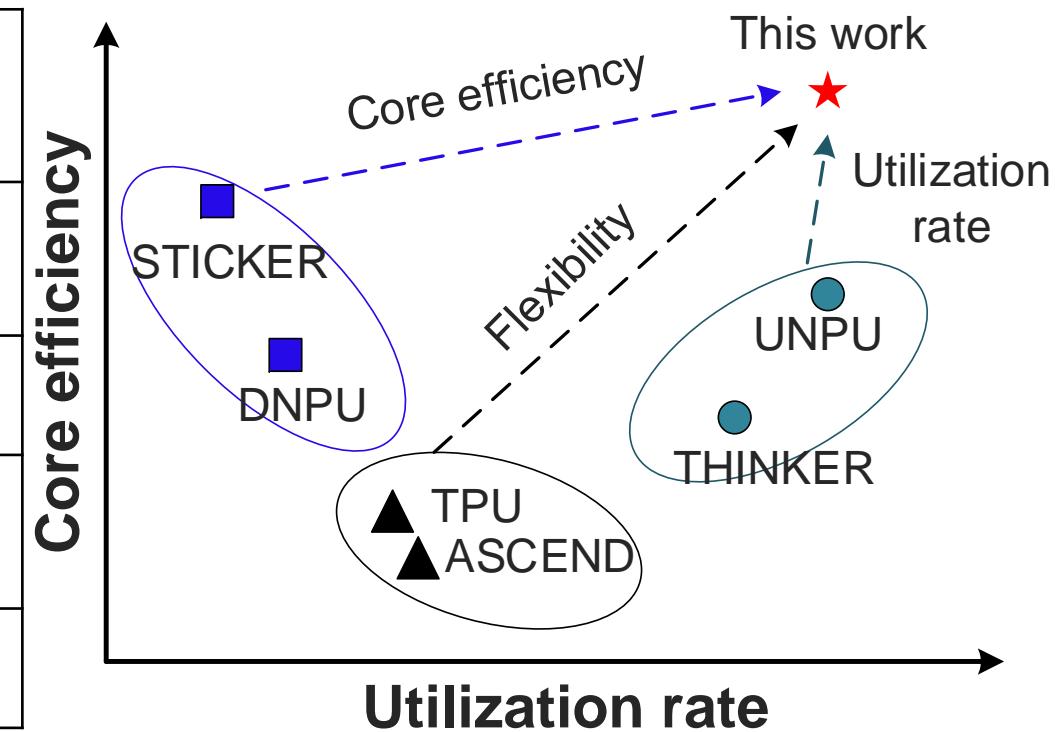


TPU, ISCA2017 92TOPS@40W
Edge TPU, 2018

Better flexibility but limited efficiency (1~2TOPS/W)

Unified Neural Network Processor

	Heteroge-neous NN	Reconfi-gurableNN	Industrial NN
Core energy /area efficiency	★ ★ ★	★ ★	★
Utilization rate	★	★ ★ ★	★ ★
Compiler /scheduling	★ ★	★	★ ★ ★
Flexibility	★ ★	★ ★	★ ★ ★

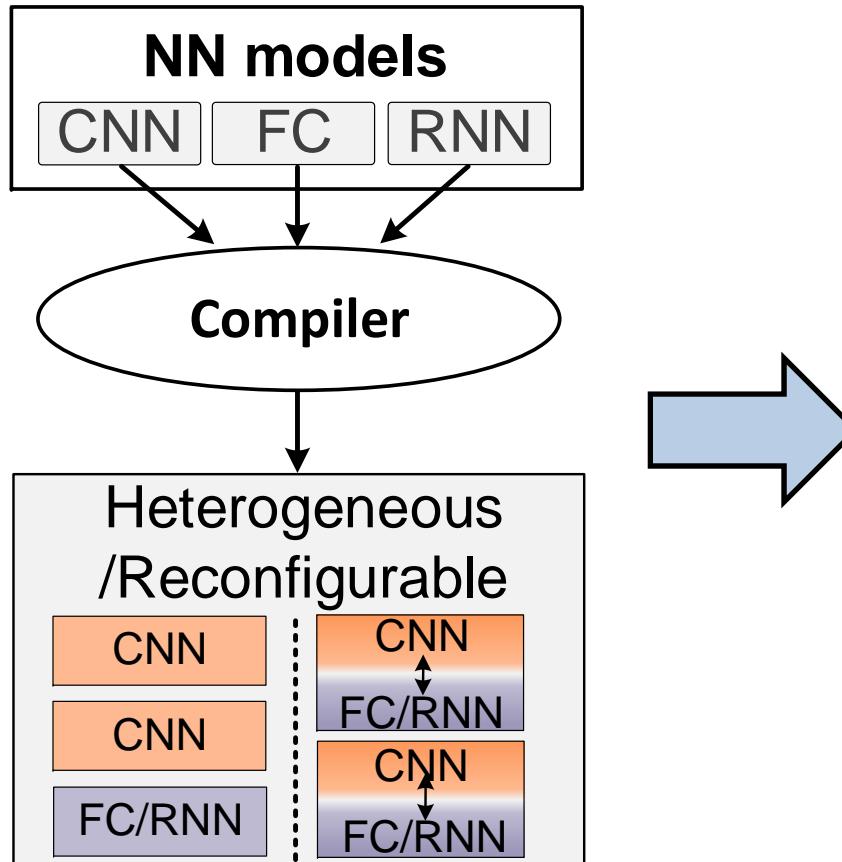


**Algorithm-hardware co-optimized solution is promising
for efficient unified NN processor**

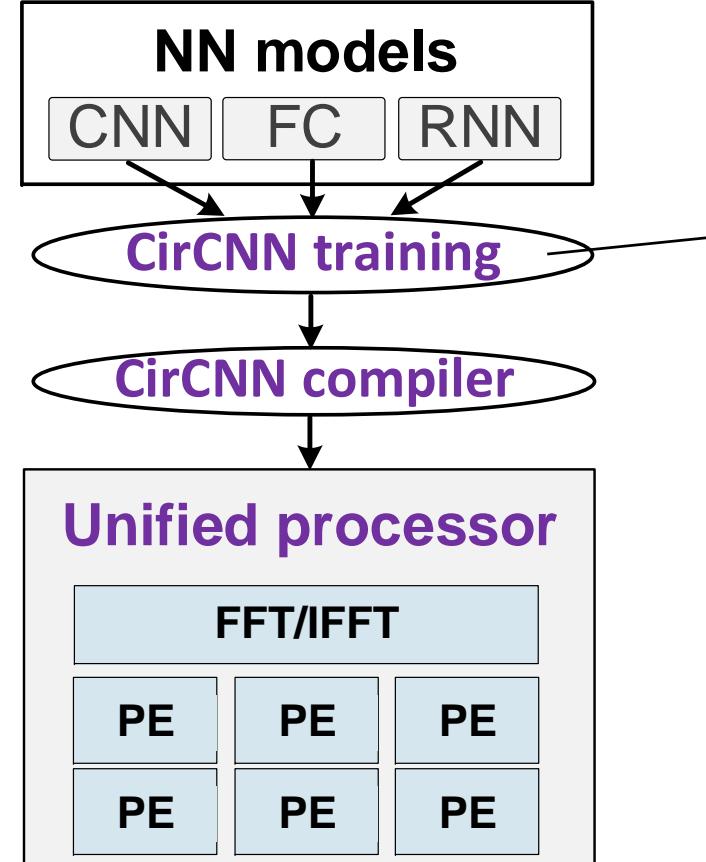
Outline

- ❑ Introduction
- ❑ Challenges of Unified Block-Circulant Processor
- ❑ Proposed Unified NN Processor
 - Global-Parallel Bit-Serial FFT Module
 - 2-D Data-Reuse Array
 - HBST TRAM
- ❑ Measurement Results
- ❑ Conclusion

CirCNN: Block-Circulant NN Workflow



Traditional workflow



CirCNN workflow

Unified workflow

- Storage reduction
- Small accuracy loss
- T-D acceleration

CirCNN Mechanism

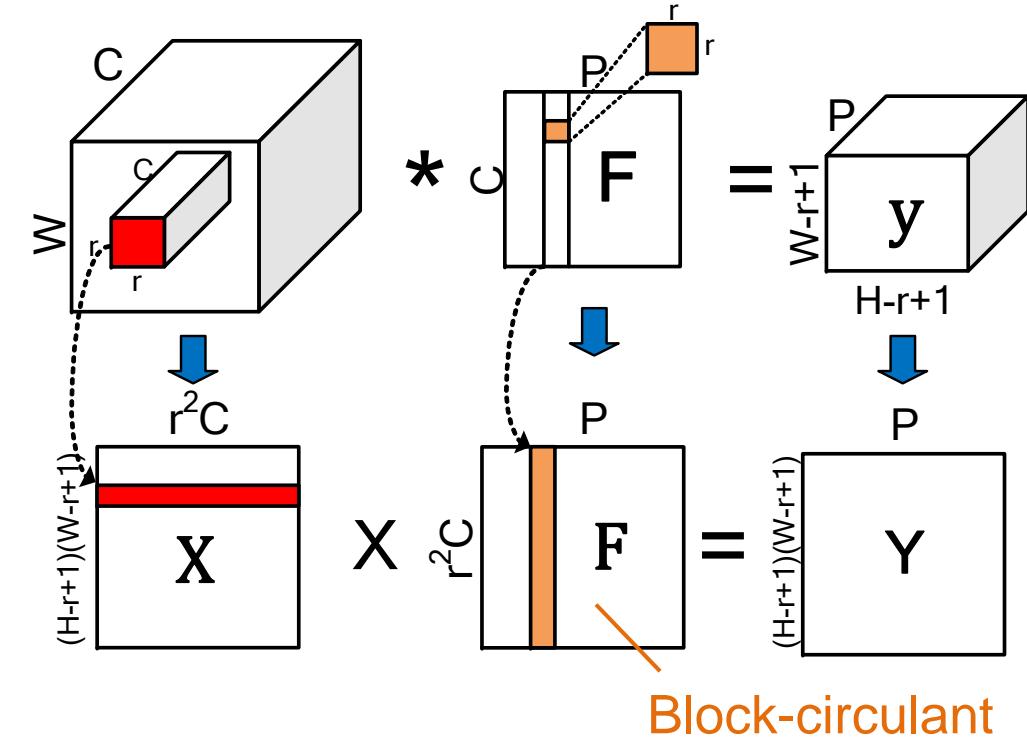
[CirCNN, Micro2017]

$$\begin{array}{c}
 \begin{matrix}
 -1.39 & 0.06 & 1.56 & 0.36 \\
 0.36 & -1.39 & 0.06 & 1.56 \\
 1.56 & 0.36 & -1.39 & 0.06 \\
 0.06 & 1.56 & 0.36 & -1.39
 \end{matrix} \times \begin{matrix} 0.36 \\ 0.43 \\ -0.12 \\ 3.42 \end{matrix} = \begin{matrix} 0.57 \\ 4.86 \\ 1.09 \\ -4.10 \end{matrix} \\
 W_{ij}
 \end{array}$$

Block-circulant

Block-circulant FC/RNN

Matrix-vector multiplication



CirCNN Training Flow

[CirCNN, Micro2017]

-1.39	0.06	1.56	0.36
0.36	-1.39	0.06	1.56
1.56	0.36	-1.39	0.06
0.06	1.56	0.36	-1.39

Circulant matrix

x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x
<hr/>			
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

Unstructured weight
(32 parameters)

Block-circulant
training



x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x
<hr/>			
x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

Block size = 4
(8 parameters)

CirCNN Training Flow

[CirCNN, Micro2017]

X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X
<hr/>			
X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X

Block-circulant training

Unstructured weight
(32 parameters)

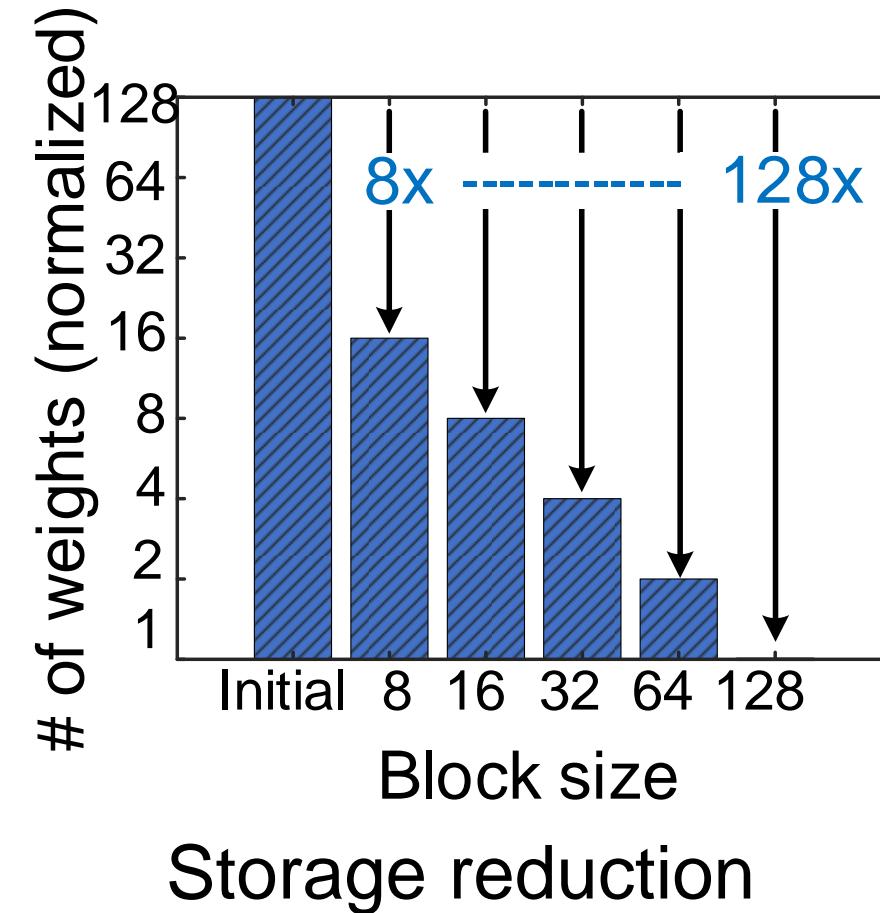
X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X
<hr/>			
X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X

Block size = 4
(8 parameters)

Input: $\frac{\partial L}{\partial \mathbf{a}}$, \mathbf{w}_{ij} 's, \mathbf{x} , p , q , k
Output: $\frac{\partial L}{\partial \mathbf{w}_{ij}}$'s, $\frac{\partial L}{\partial \mathbf{x}}$
Initialize $\frac{\partial L}{\partial \mathbf{w}_{ij}}$'s and $\frac{\partial L}{\partial \mathbf{x}}$ with zeros.
for $i \leftarrow 1$ **until** p **do**
 for $j \leftarrow 1$ **until** q **do**
 $\frac{\partial L}{\partial \mathbf{w}_{ij}} \leftarrow \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{x}'_j))$
 $\frac{\partial L}{\partial \mathbf{x}_j} \leftarrow \frac{\partial L}{\partial \mathbf{x}_j} + \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{w}_{ij}))$
 end
end
return $\frac{\partial L}{\partial \mathbf{w}_{ij}}$'s, $\frac{\partial L}{\partial \mathbf{x}}$

Circulant back-propagation

Storage & Accuracy After Training



Model	Tiny YOLO	LSTM	FC
Dataset	DataDJI	TIMIT	MNIST
Layer type	CNN	RNN (LSTM)	FC
Block size	16	32	16
MAC bit-precision	5	5	4
Accuracy loss	1.30%	3%	1.01% 1.93%
			1.00% 1.50%

Transform-Domain Acceleration

Unified CNN/FC/RNN flow

$$\begin{array}{|c|c|c|c|} \hline -1.39 & 0.06 & 1.56 & 0.36 \\ \hline 0.36 & -1.39 & 0.06 & 1.56 \\ \hline 1.56 & 0.36 & -1.39 & 0.06 \\ \hline 0.06 & 1.56 & 0.36 & -1.39 \\ \hline \end{array} \times \begin{matrix} 0.36 \\ 0.43 \\ -0.12 \\ 3.42 \end{matrix} = \begin{matrix} 0.57 \\ 4.86 \\ 1.09 \\ 4.10 \end{matrix}$$

W_{ij} x y

Transform-domain acceleration

1-D convolution

$$y[n] = \sum_{m=0}^{M-1} w_i[n-m] \cdot x[m]$$

Time domain

Transpose domain

$$Y[k] = W[k] \cdot X[k]$$

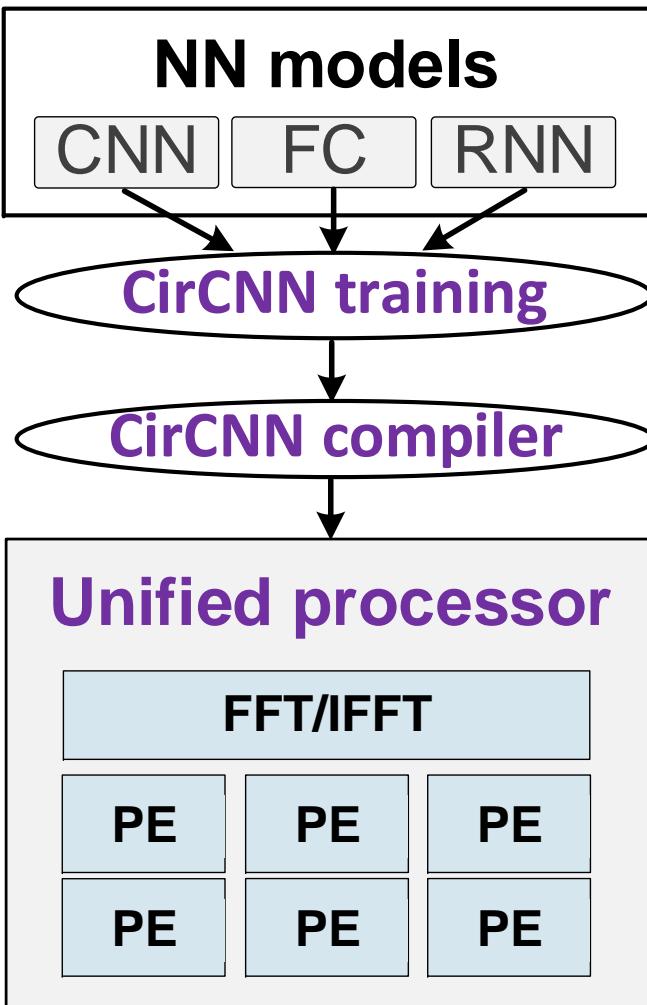
$O(n^2)$

$O(n \log n)$

Diagram illustrating the transform-domain acceleration for 1-D convolution. It shows the Time domain (y[n]) and Transpose domain (Y[k]) representations, along with their respective FFT/IFFT operations and the formula for the Transpose domain multiplication.

n (block-size)	8	16	32	64	128
$O(n^2)/O(n \log n)$	2.7x	4.0x	6.4x	10.7x	18.3x

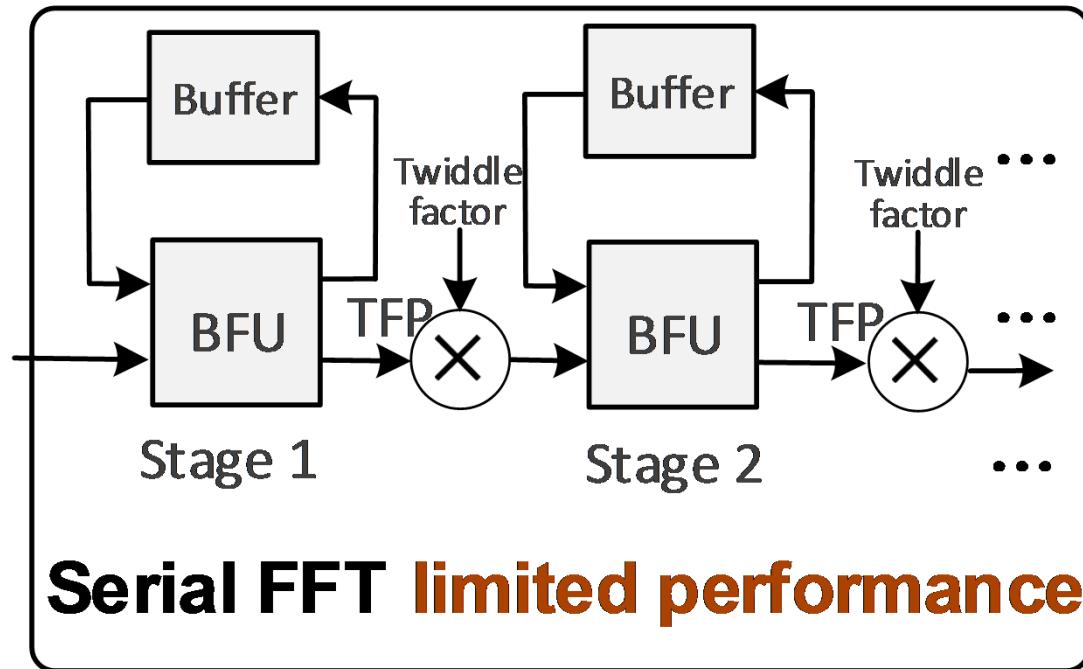
CirCNN: Block-Circulant NN



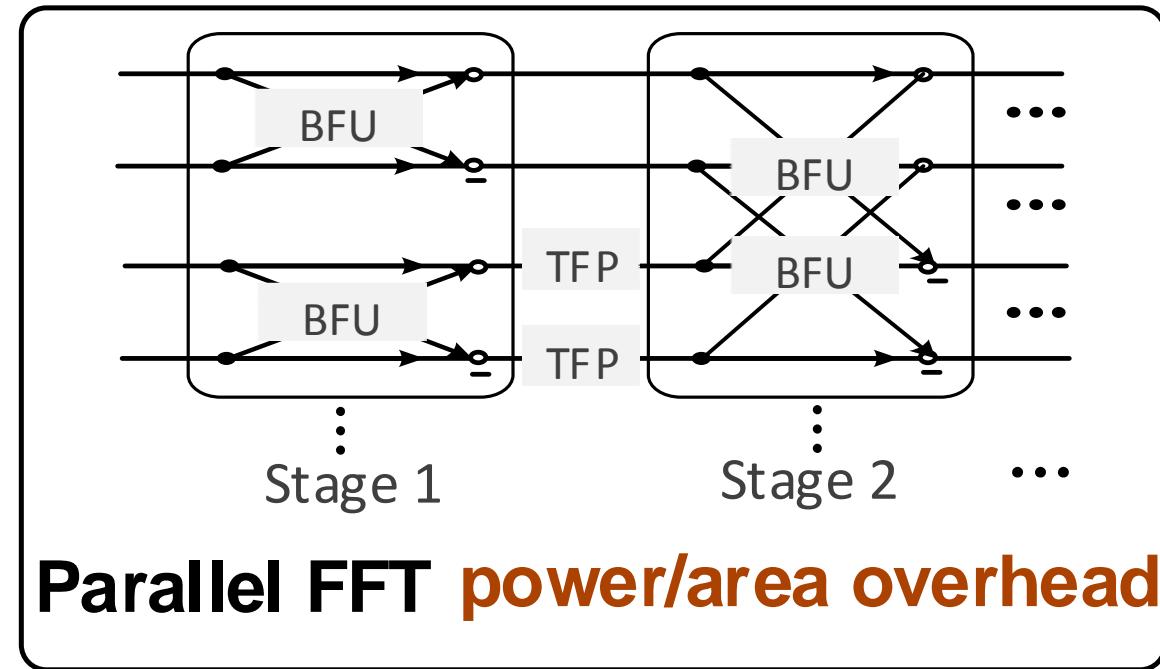
- **Unified workflow**
- **Storage reduction**
- **Small accuracy loss**
- **T-D acceleration**

Main operations in CirCNN processor
Fast Fourier Transformation (FFT)
Element-wise production

Challenge 1: FFT Overhead



Serial FFT limited performance



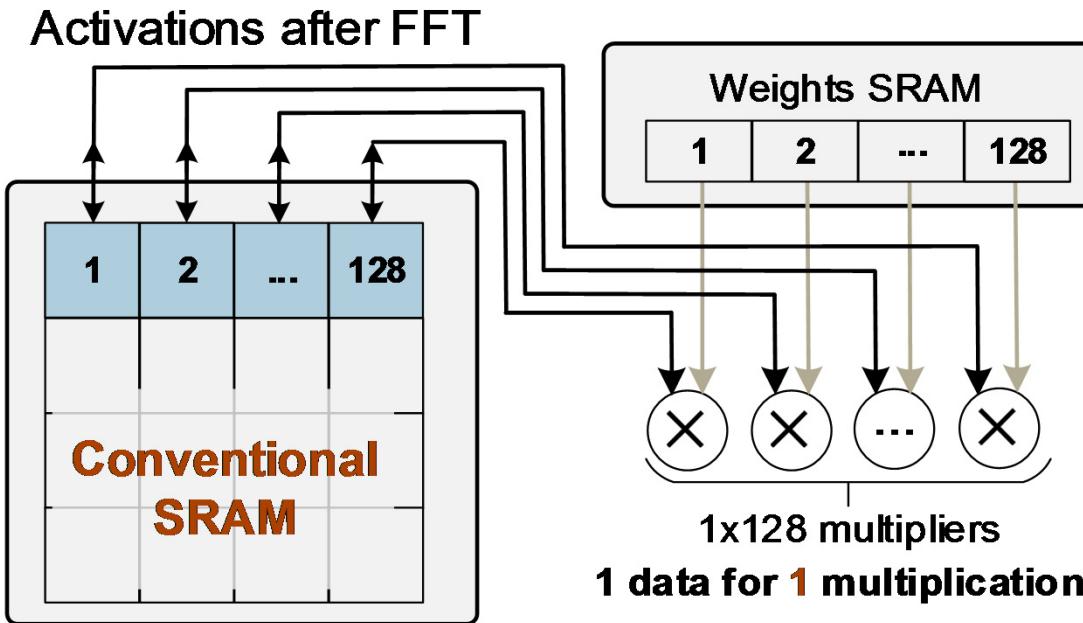
Parallel FFT power/area overhead

Global-parallel bit-serial FFT

Challenge 2: Data Reuse Problem

$$Y[k] = W[k] \cdot X[k]$$

Element-wise multiplication



Conventional SRAM based
MAC array

No data reuse

High bandwidth

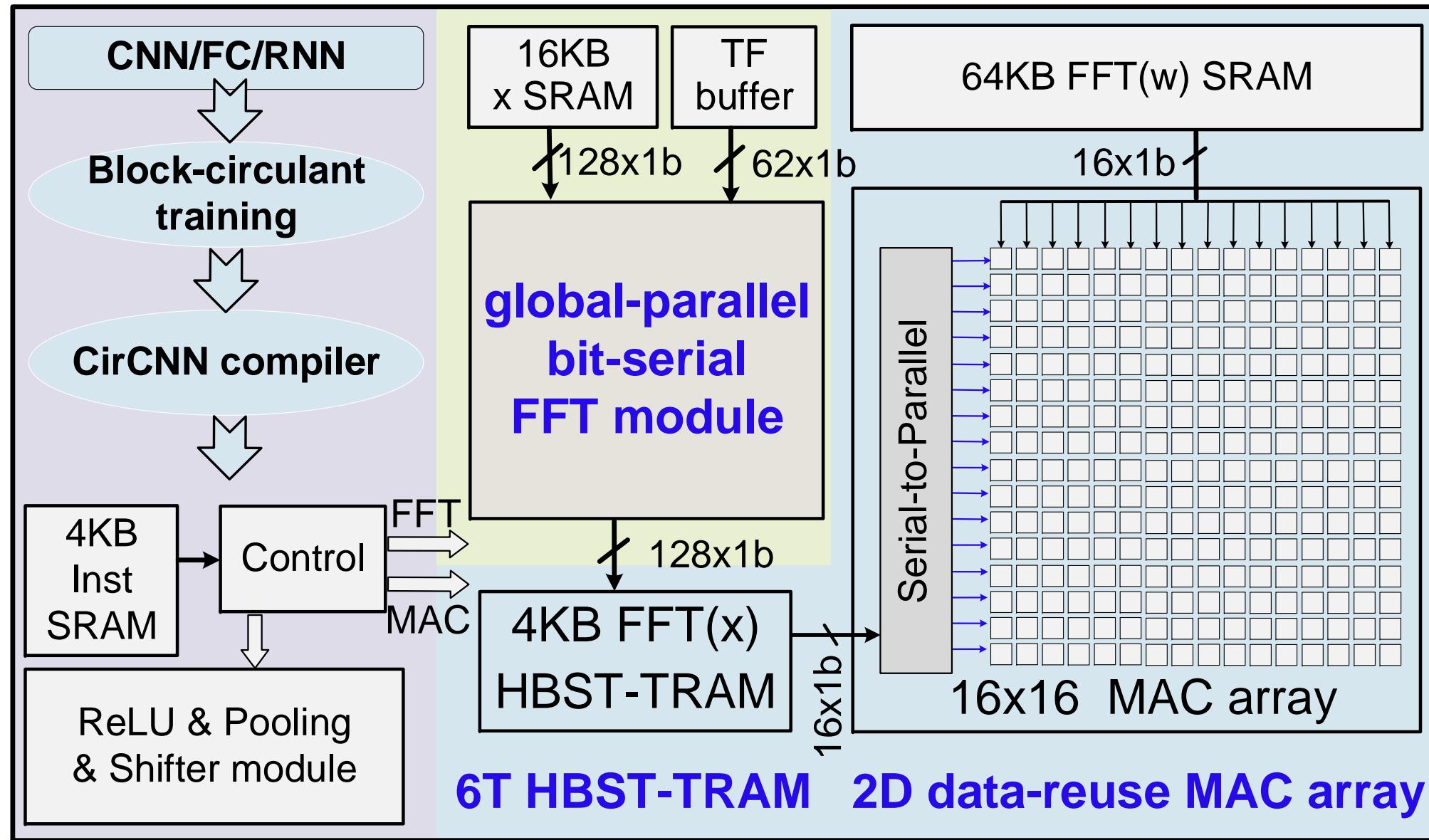
High memory access

HBST-TRAM based
2-D data-reuse array

Outline

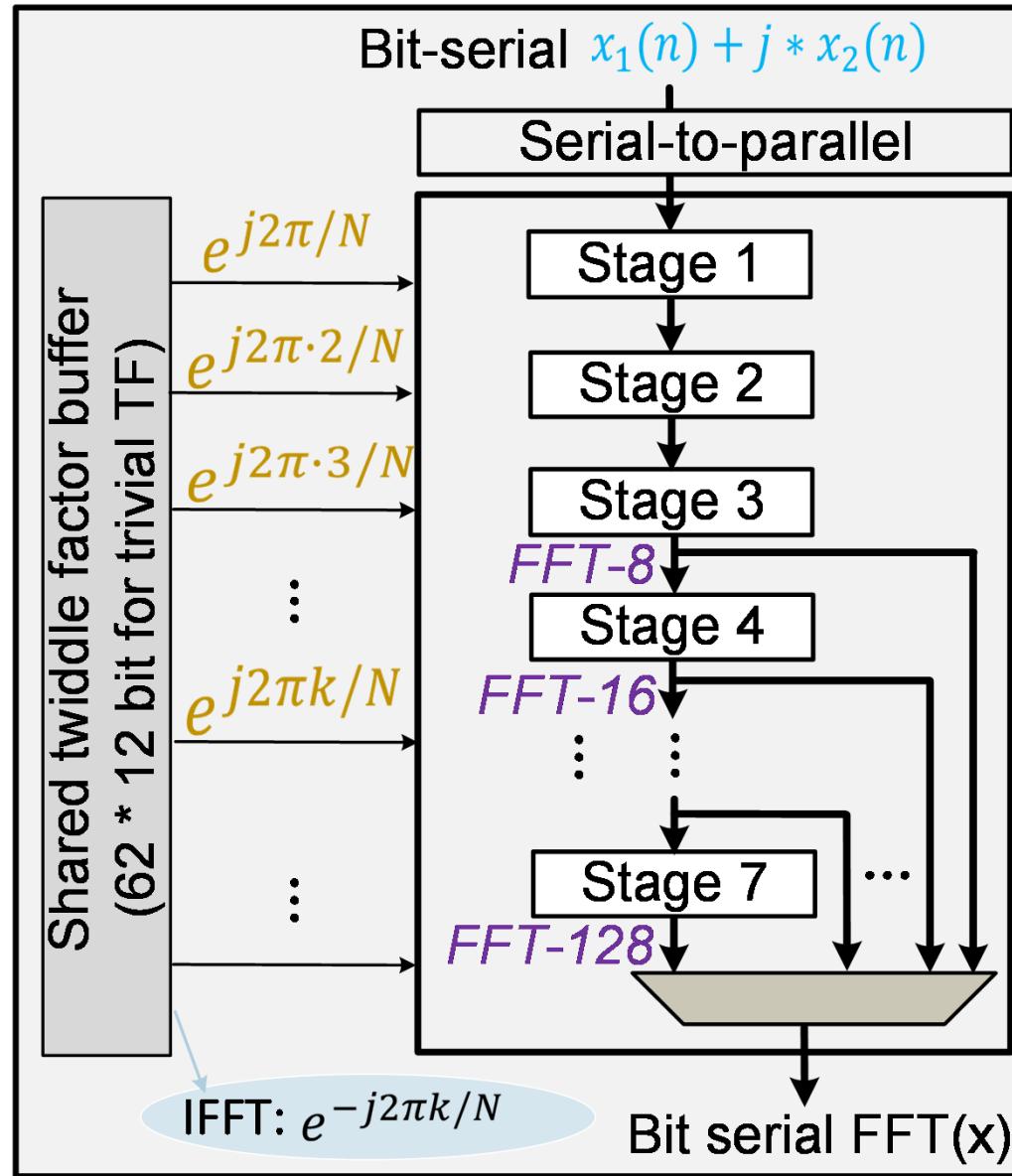
- ❑ Introduction
- ❑ Challenges of Unified Block-Circulant Processor
- ❑ Proposed Unified NN Processor
 - Global-Parallel Bit-Serial FFT Module
 - 2-D Data-Reuse Array
 - HBST TRAM
- ❑ Measurement Results
- ❑ Conclusion

Overall Architecture of STICKER-T

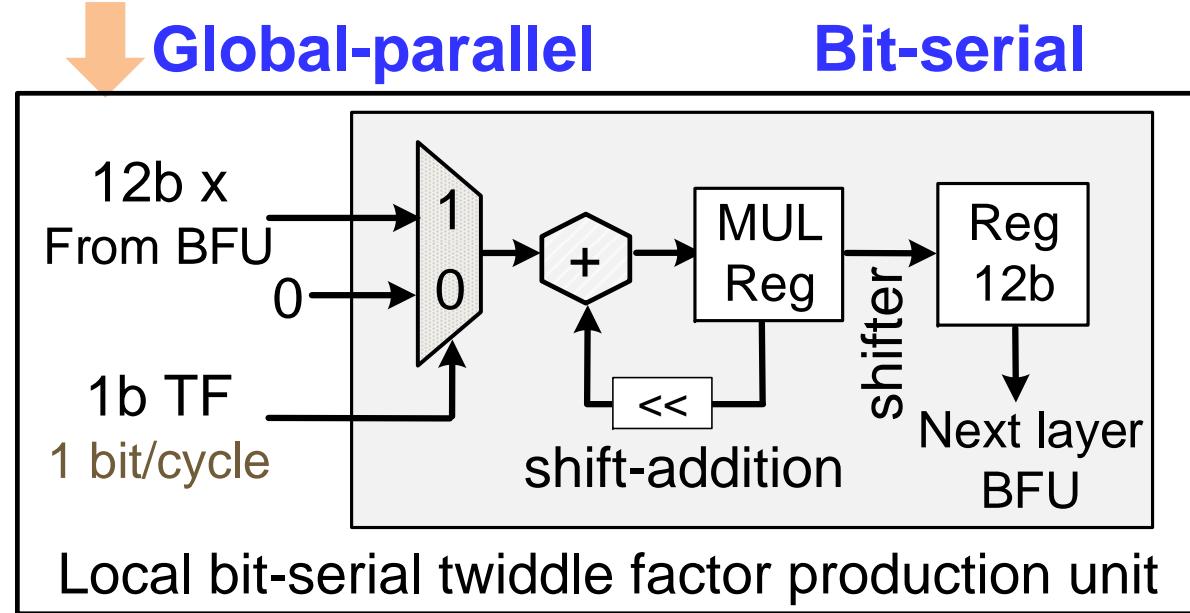
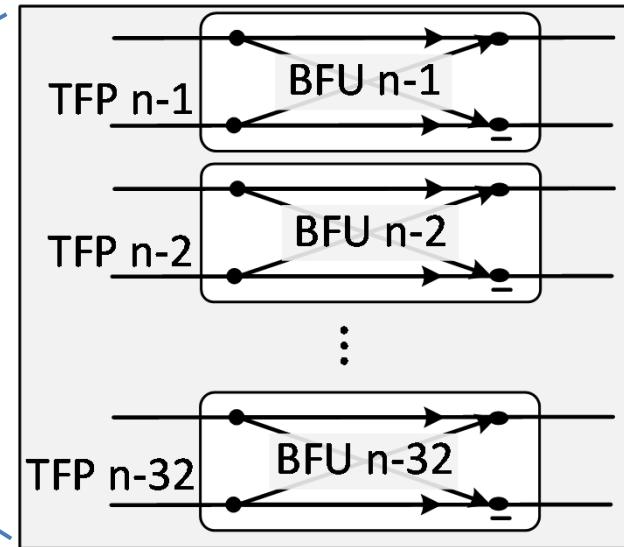
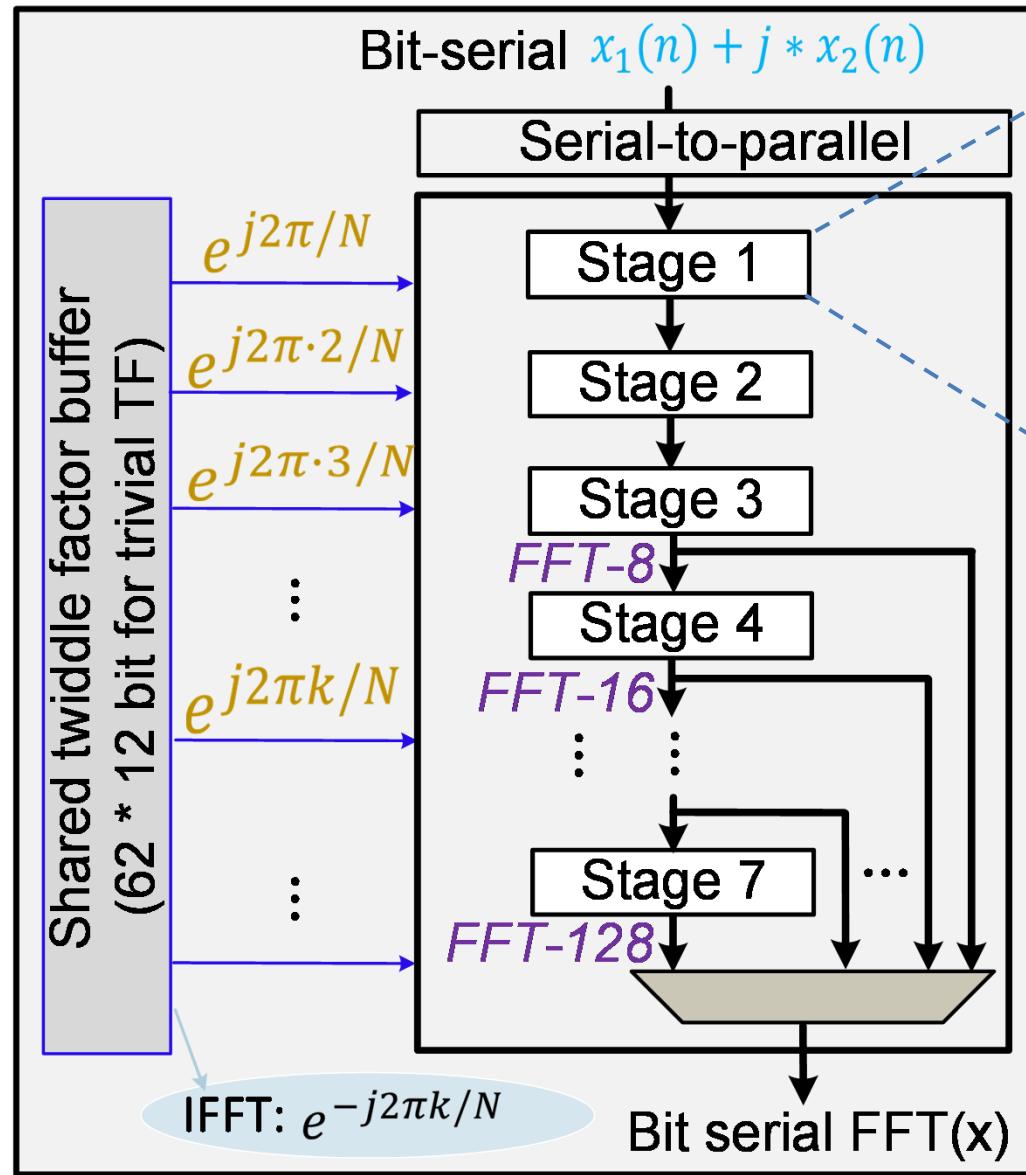


Global-Parallel Bit-Serial FFT

- Shared twiddle-factor (TF) buffer
- Skip non-trivial TFs
- Real activation feature



Global-Parallel Bit-Serial FFT

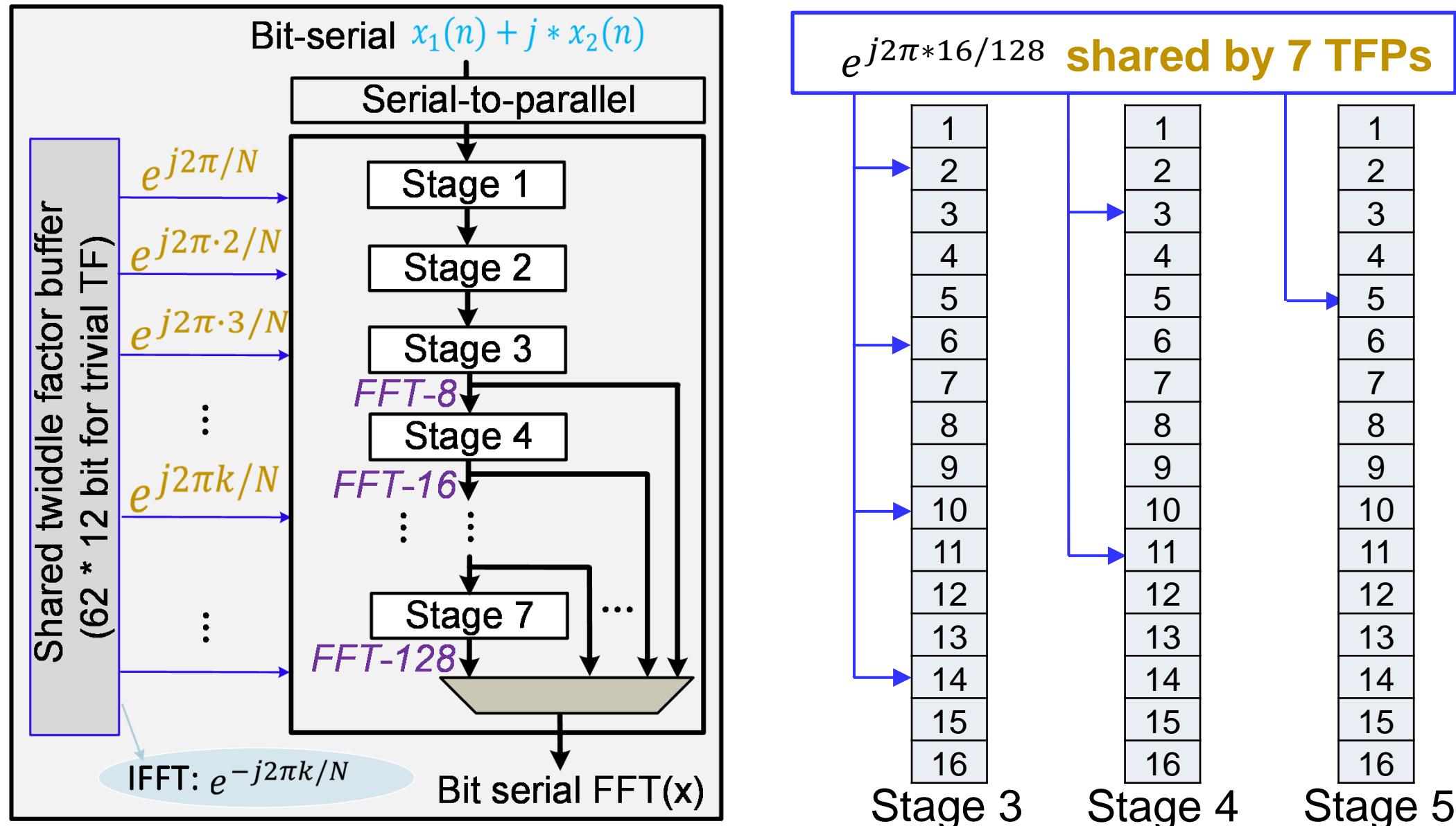


TFP: twiddle factor production unit
BFU: butterfly unit

Global-parallel

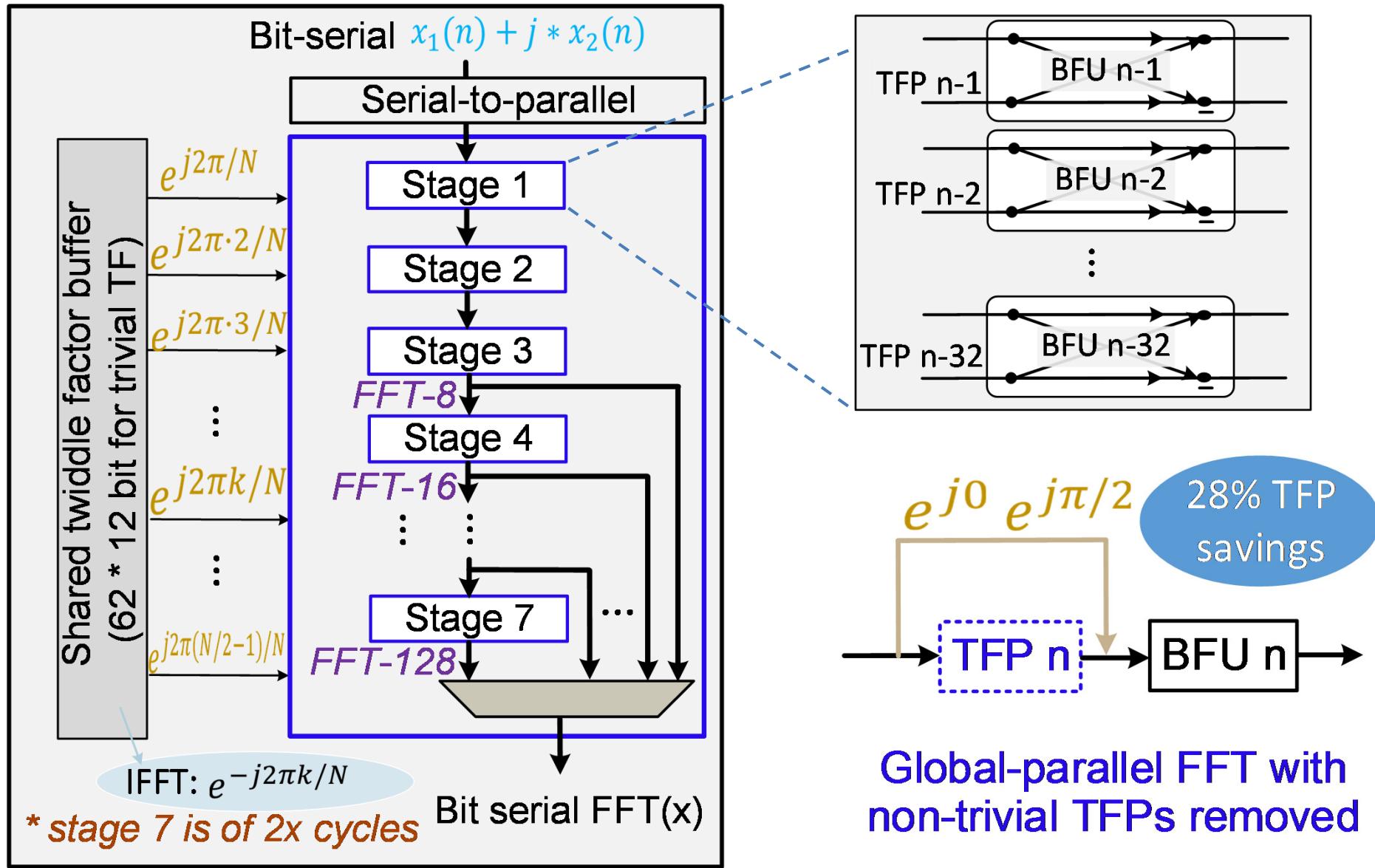
Bit-serial

Shared Twiddle-Factor (TF) Buffer



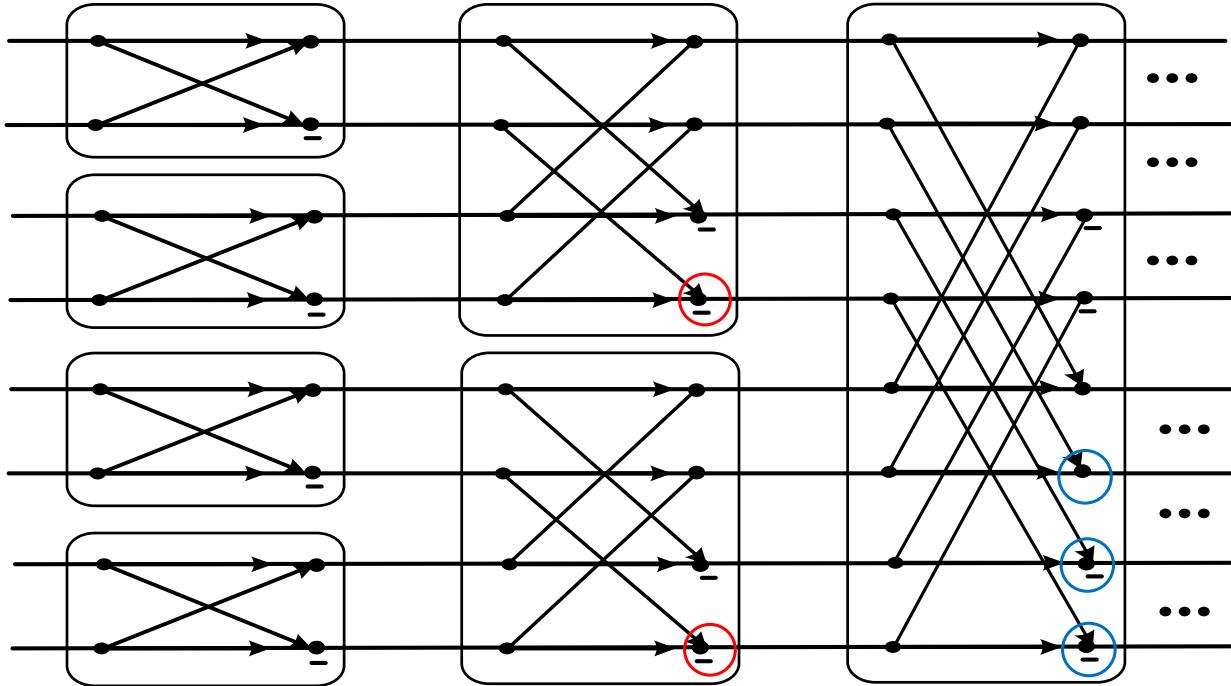
7.5: A 65nm 0.39-to-140.3TOPS/W 1-to-12b Unified Neural-Network Processor Using Block-Circulant-Enabled Transpose-Domain Acceleration with
8.1 X Higher TOPS/mm² and 6T HBST-TRAM-Based 2D Data-Reuse Architecture

Skip Non-Trivial TFs



Real Activation Feature Based Optimization

Activations $\in \mathbb{R}$



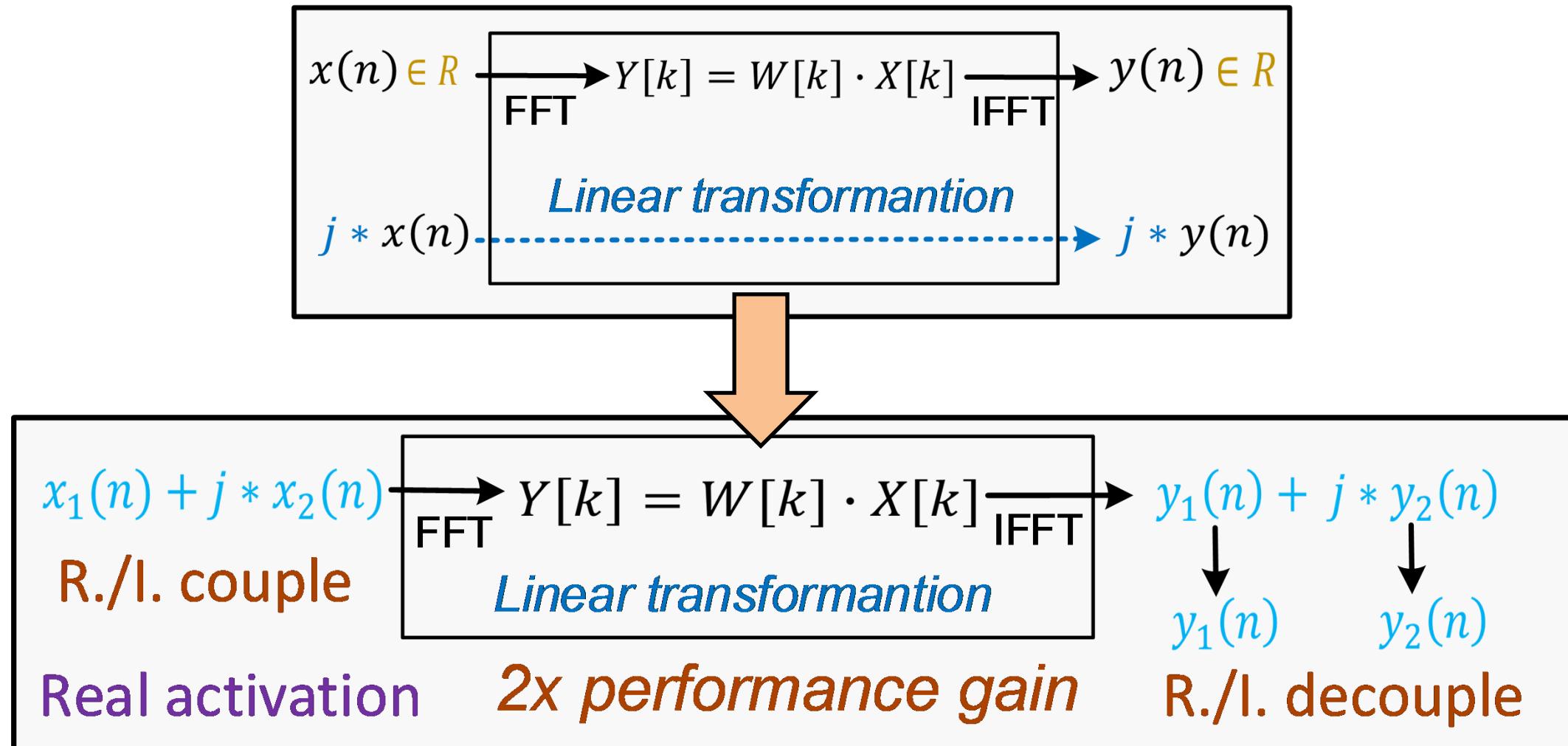
$$X[k] = X^*[N - k]$$

Red circle: No need to be FFT-4
Blue circle: calculated in FFT-8

Hardware-unfriendly

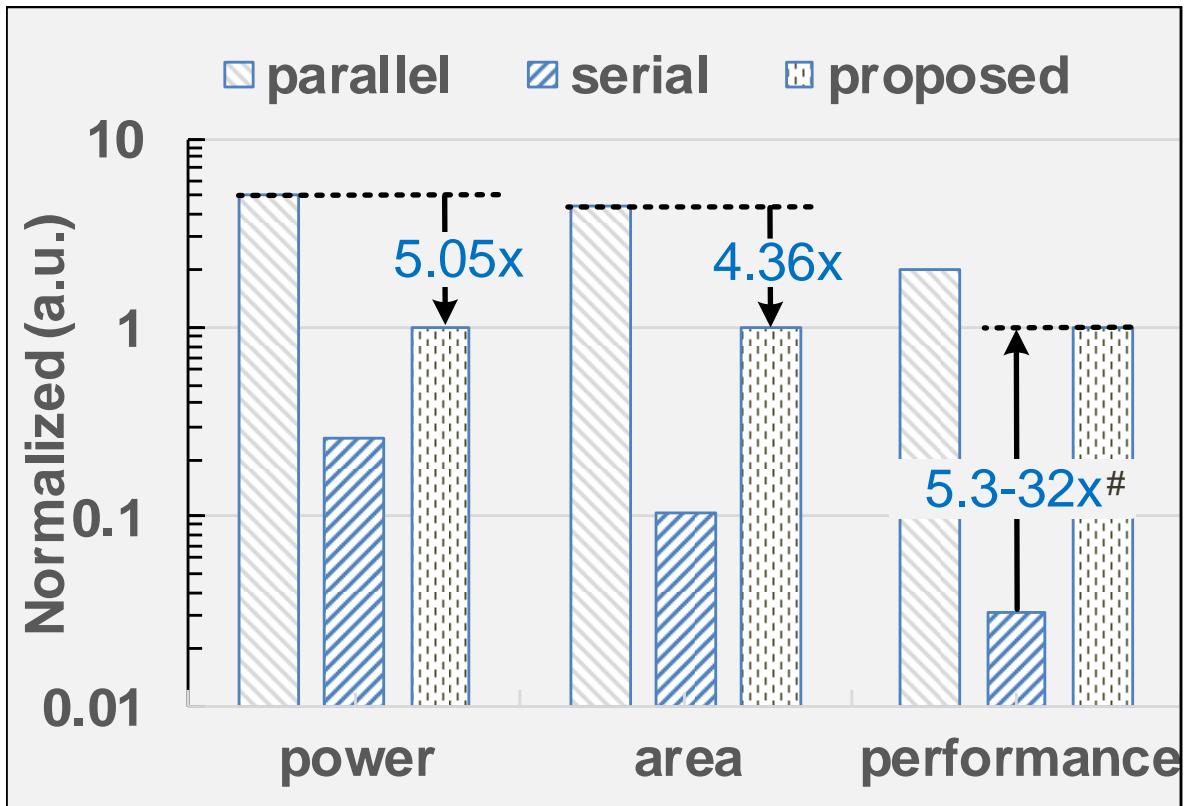
Irregular partial utilization of real activation

Real Activation Feature Based Optimization



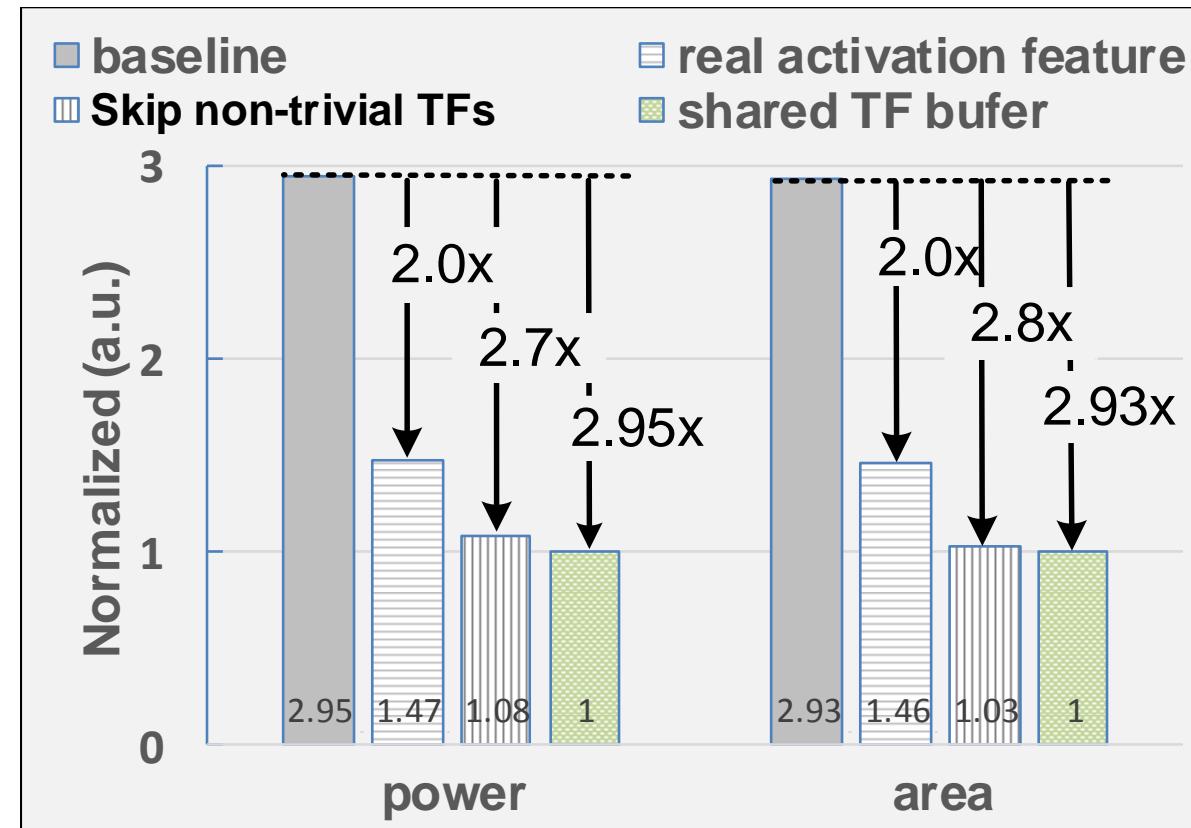
Full utilization of real activation

FFT Optimizing Results



Throughput varies for 1-12b FFT.

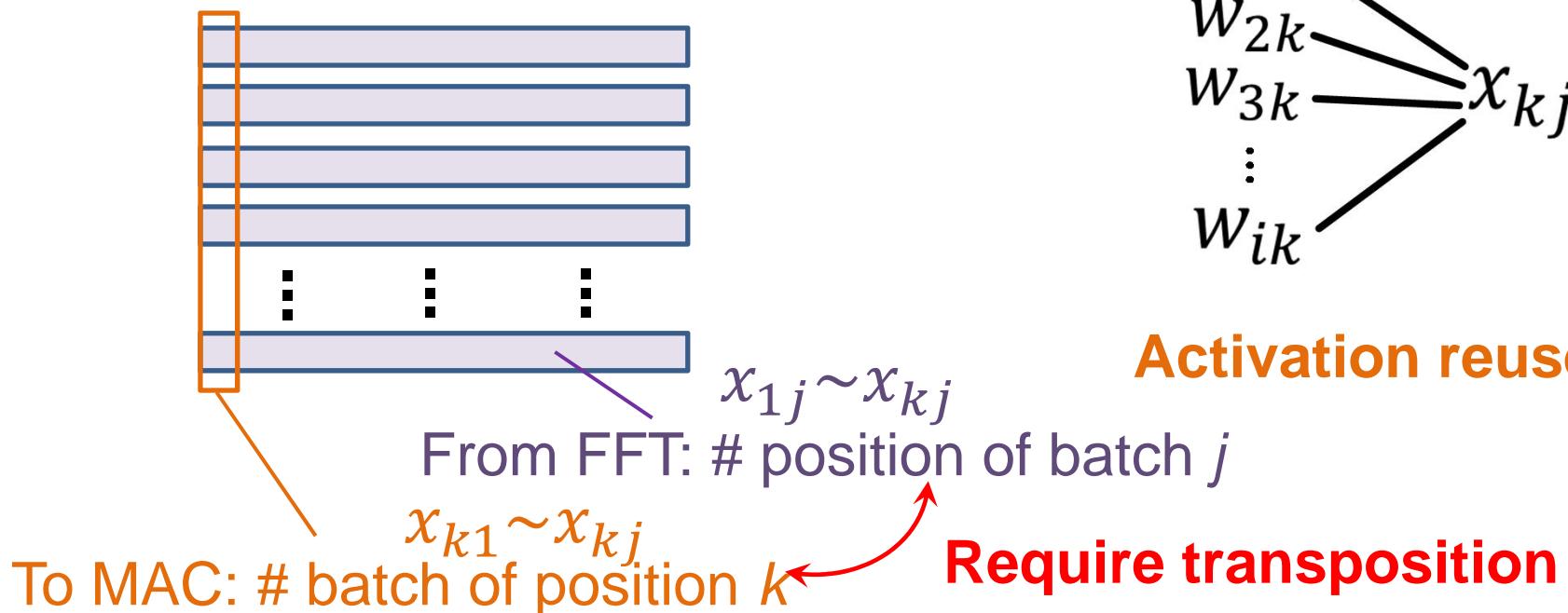
Power/area/performance improvement



Power/area reduction

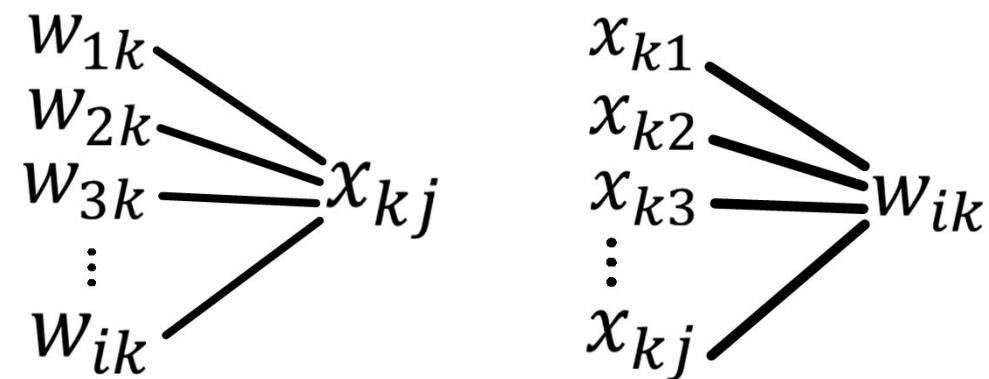
2-D Data-Reuse MAC Array

- 2-D data-reuse manner
- TRAM transformation for data reuse
- Bit-serial PE



$$FFT(y_{ij}) = \sum_{k=1}^n FFT(w_{ik}) \odot FFT(x_{kj})$$

i, k : block row/col, j : activation batch.

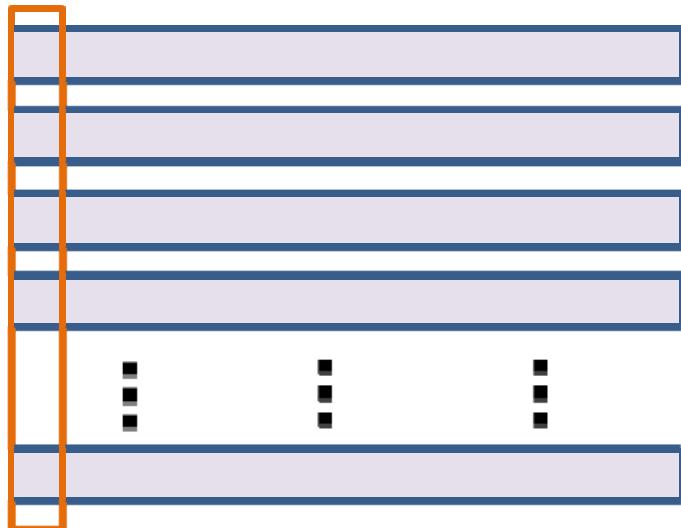


Activation reuse

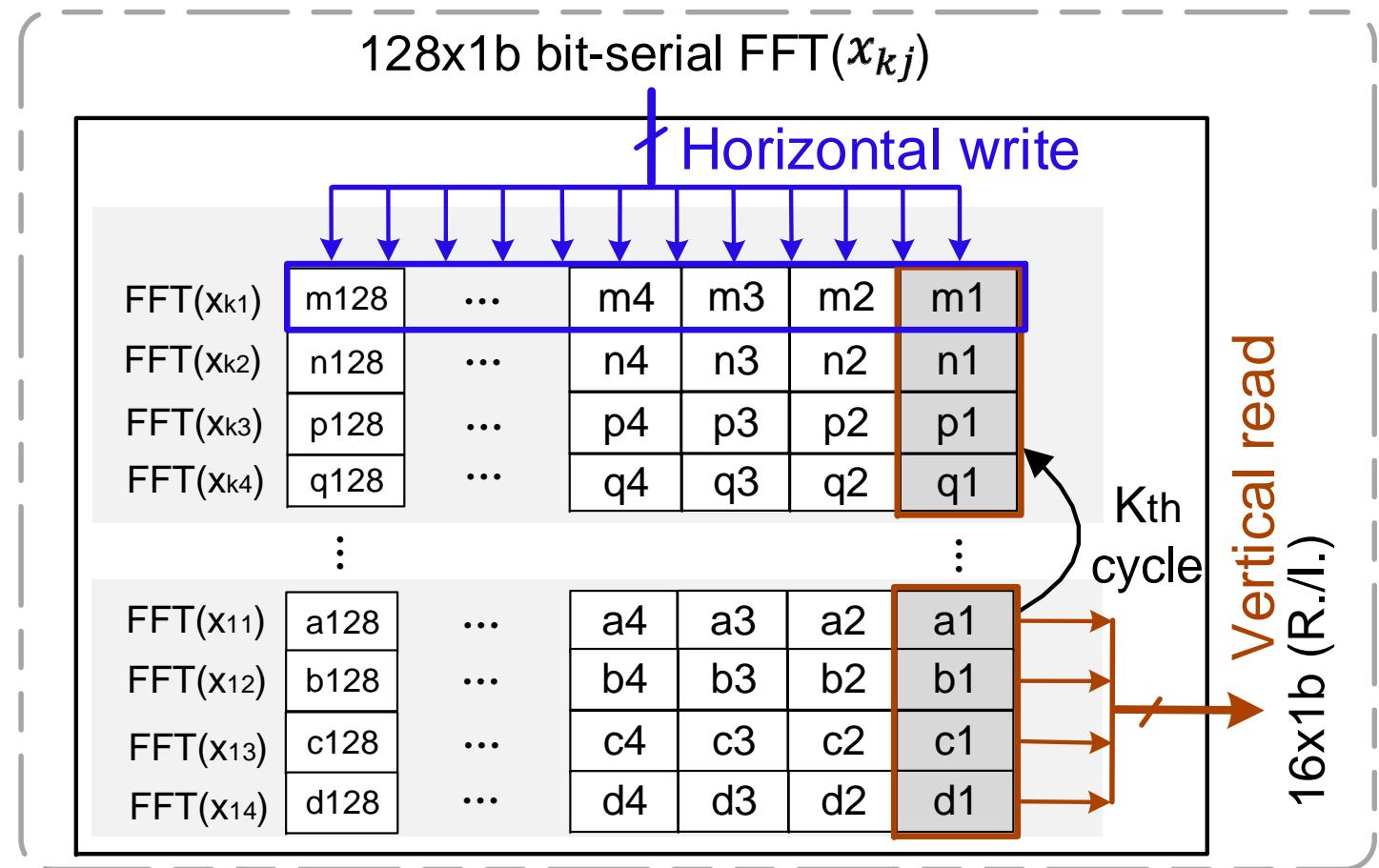
Weight reuse

2-D Data-Reuse MAC Array

- 2-D data-reuse manner
- TRAM transformation for data reuse
- Bit-serial PE

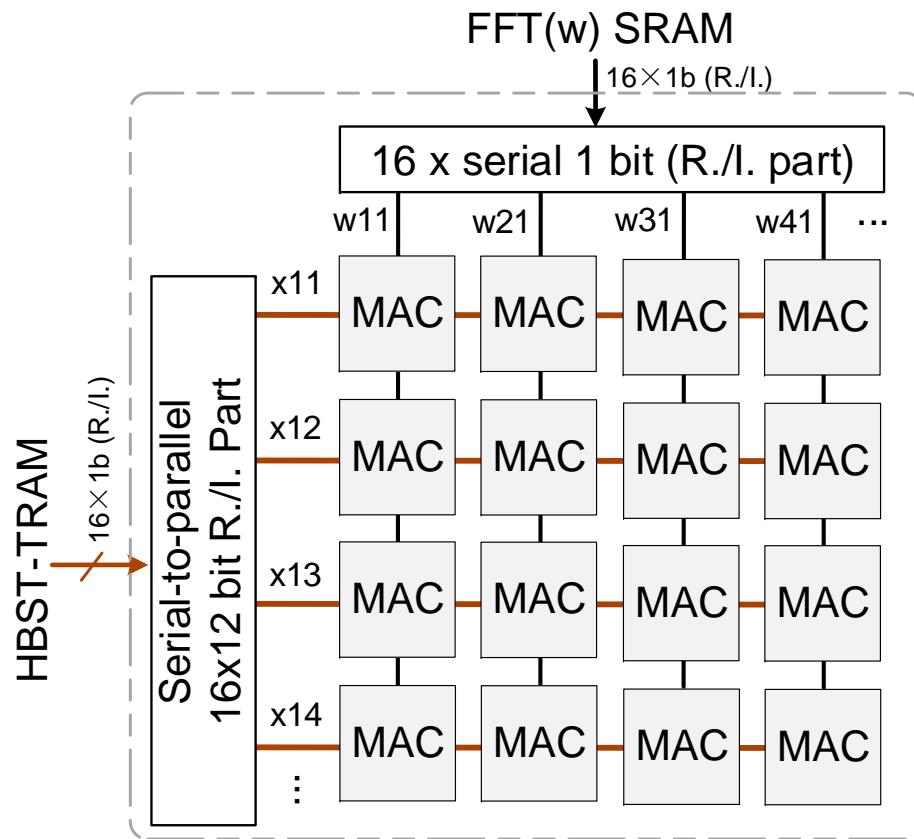


Data reuse conflict

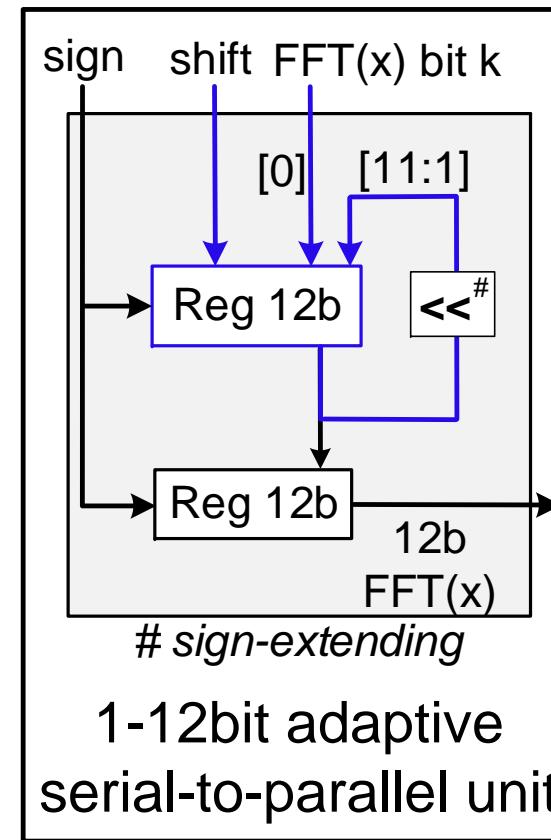


Transpose SRAM (TRAM)

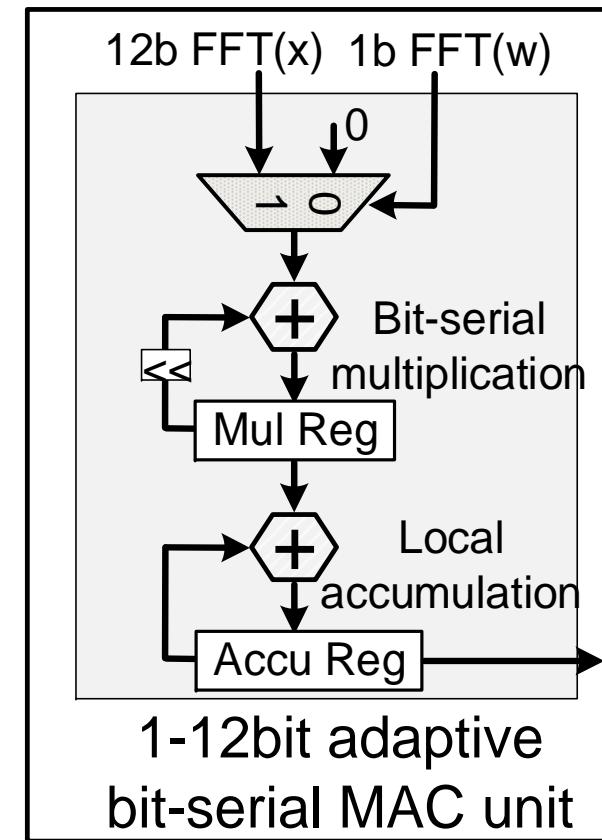
2-D Data-Reuse MAC Array



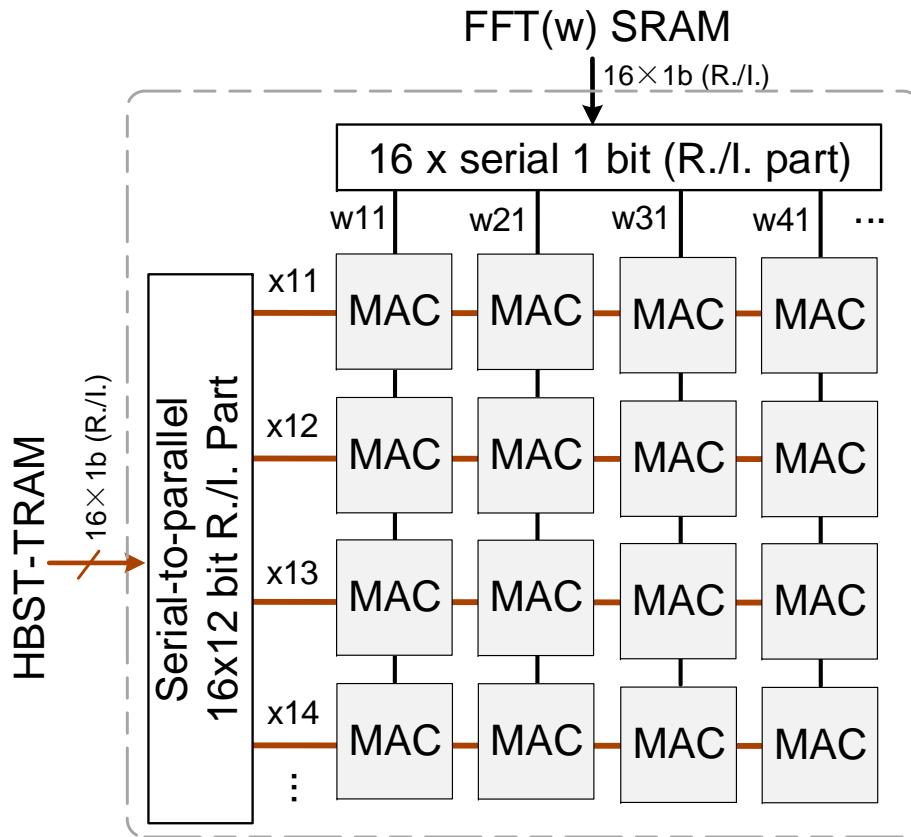
16x16 PE array



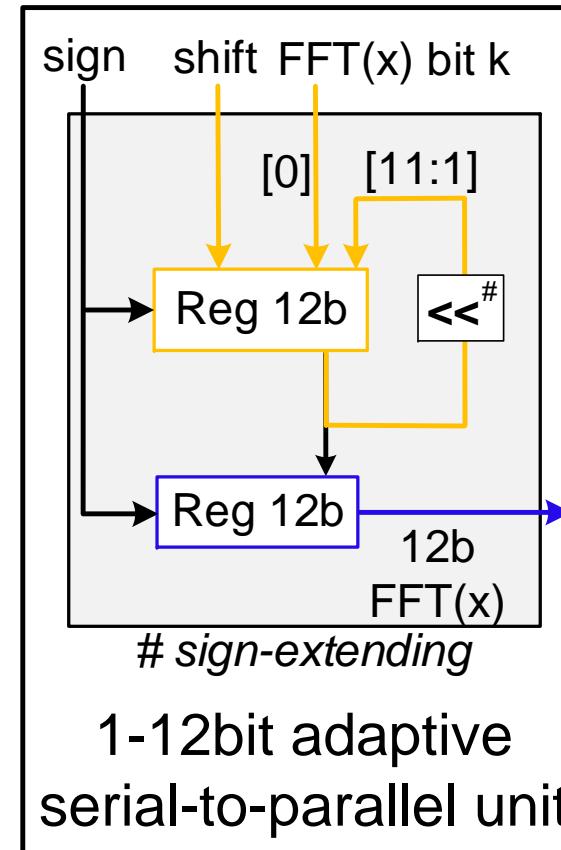
Bit-serial PE



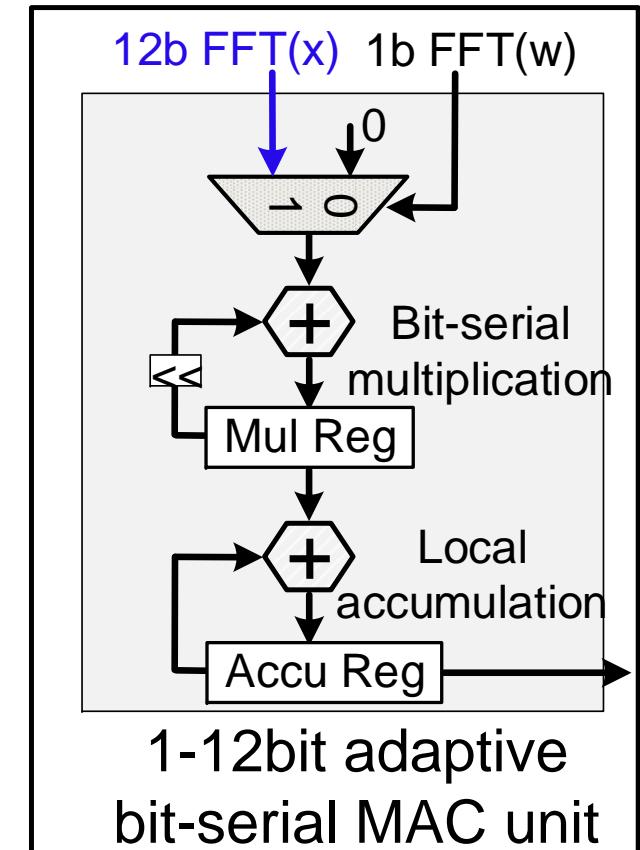
2-D Data-Reuse MAC Array



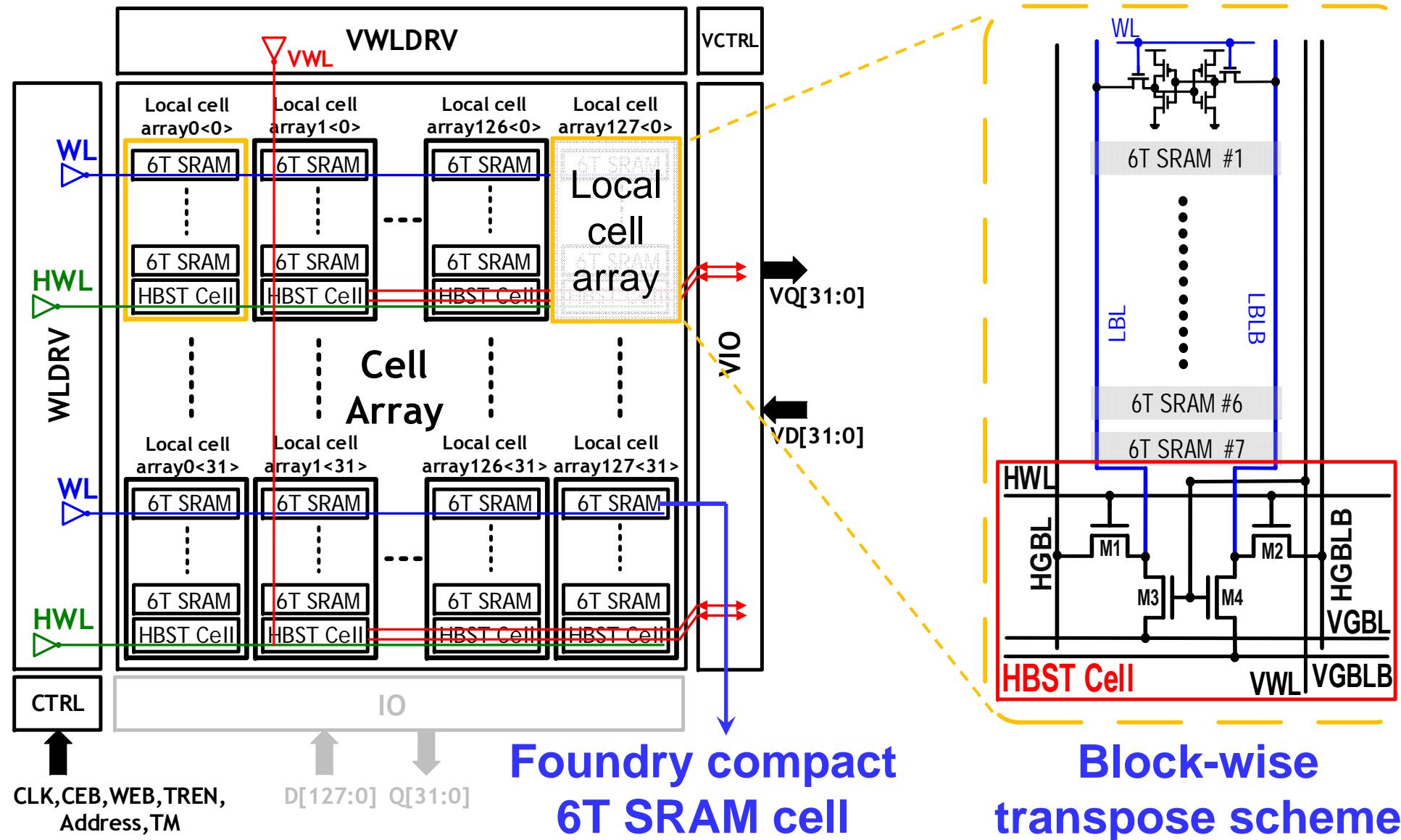
16x16 PE array



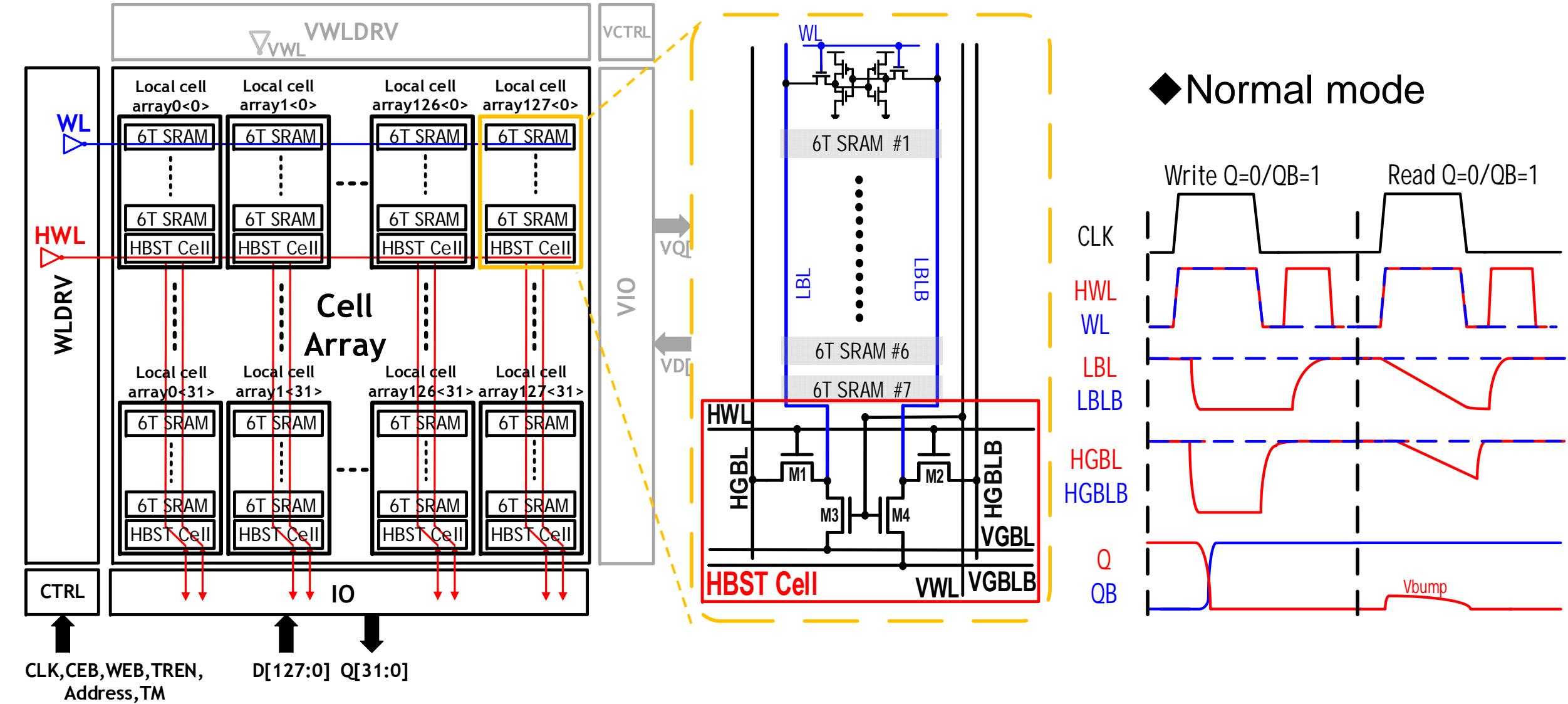
Bit-serial PE



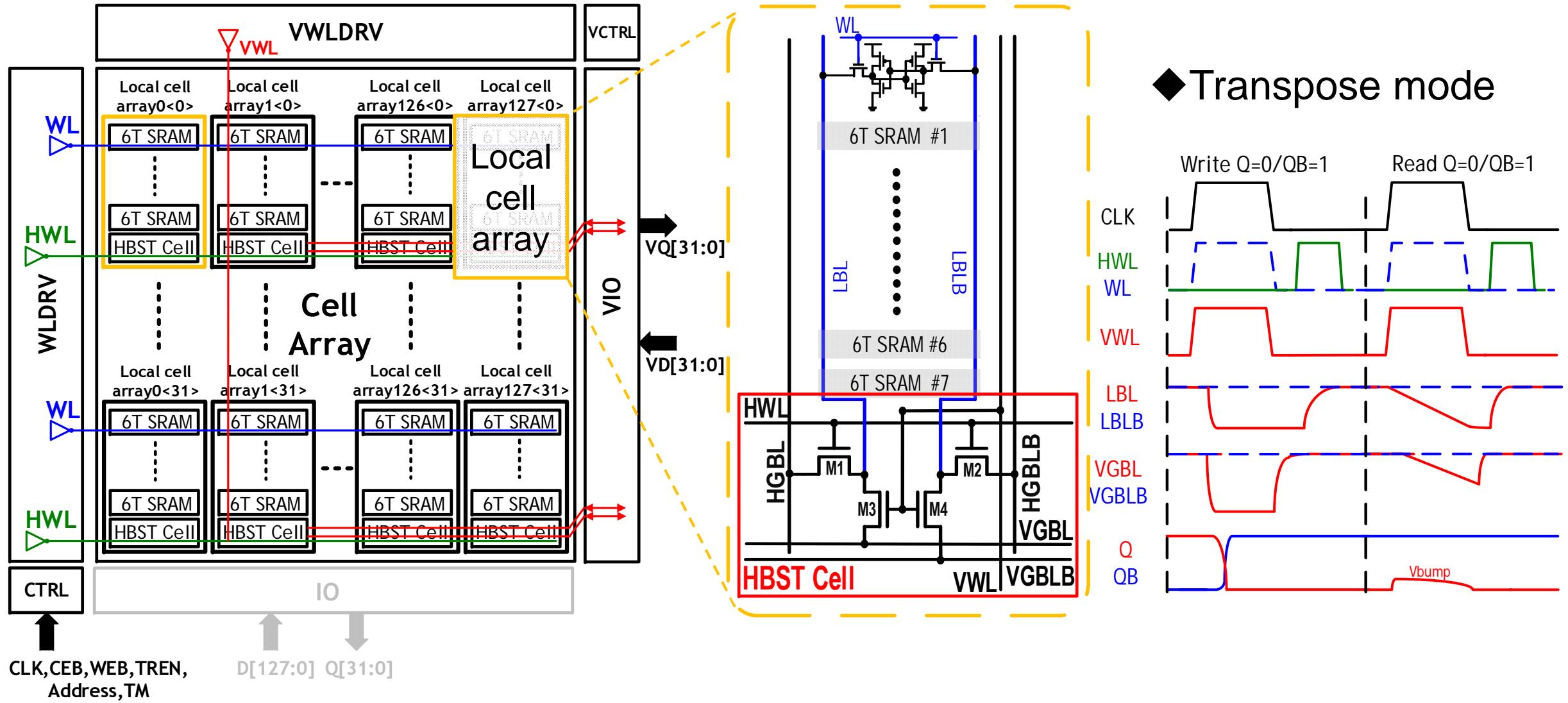
Hierarchical-Bitline-Switching (HBST) TRAM



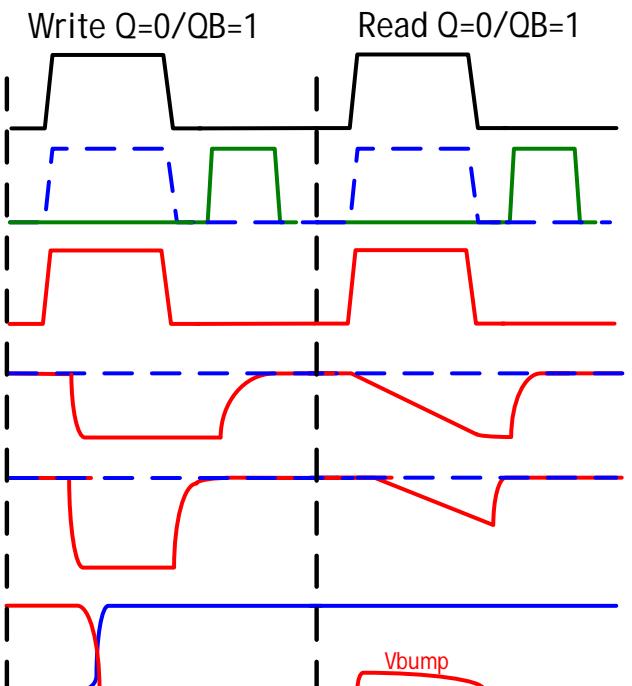
HBST-TRAM Normal Mode R/W



HBST-TRAM Transpose Mode R/W



◆ Transpose mode



HBST-TRAM Comparison

	This work	ISSCC 2017	CICC 2011
# of cell(block) transistors	6T (52T)	7T (56T)	8T (64T)
Transpose scheme	Block-wise	Cell-wise	Cell-wise
Transpose function	Read/Write	Read	Read/Write
Array size ^{*1}	1.45x (compact rule)	2.98x (logic rule)	5.9x (logic rule)
Delay ^{*2}	1x	2.09x	1.11x
Energy ^{*3}	1x	2.14x	1.29x

^{*1} Conventional compact 6T=1x

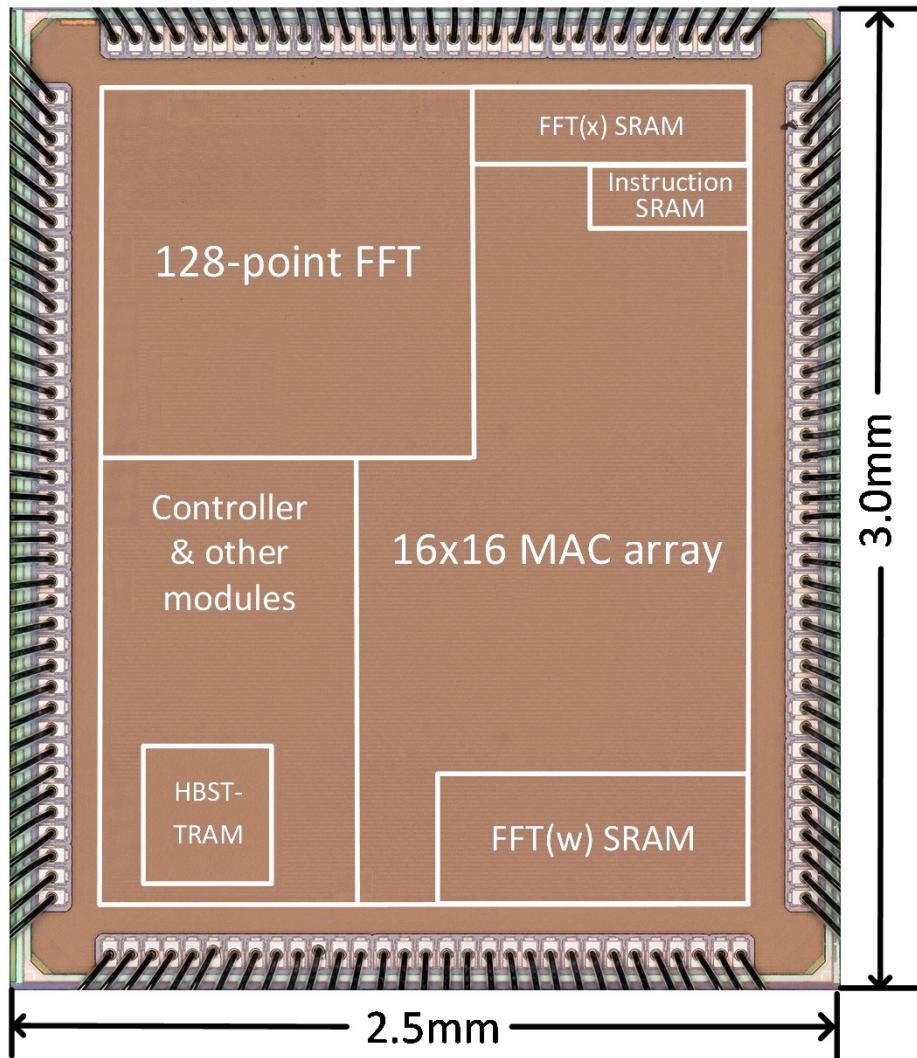
^{*2} at VDD=nominal, TT corner, 25°C, 32Kb macro, avg. of normal and transpose mode

^{*3} at VDD=nominal, FF corner, 85°C, 32Kb macro, avg. of normal and transpose mode

Outline

- ❑ Introduction
- ❑ Challenges of Unified Block-Circulant Processor
- ❑ Proposed Unified NN Processor
 - Global-Parallel Bit-Serial FFT Module
 - 2-D Data-Reuse Array
 - HBST TRAM
- ❑ Measurement Results
- ❑ Conclusion

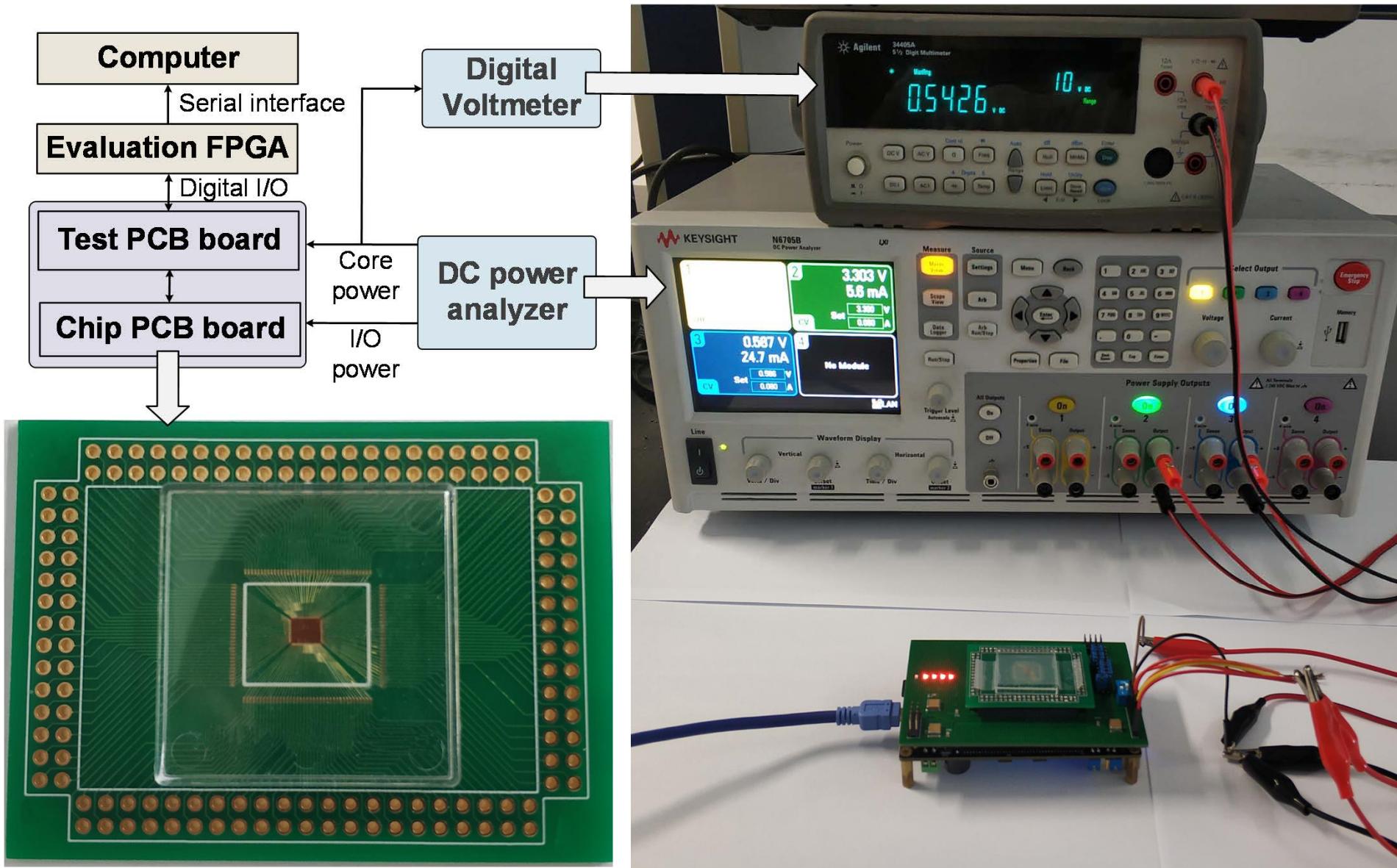
Chip Micrograph & Summary Table



Technology	TSMC 65nm GP
Chip Area	3.0mmx2.5mm
Core Area	2.46mmx1.96mm
Support NN type	CNN+FC+RNN
Weight Bit-precision	1-12 bit
Activation Bit-precision	1-12 bit
Total SRAM	100kB
HBST-TRAM	4kB
Supply Voltage	0.54-1.15V
Frequency	25-200MHz
Power	13.3-339.2mW
Performance	0.13-14.9TOPS
Energy Efficiency ¹⁾	0.39-140.3TOPS/W

1) minimum @1.15V,200MHz,12bit, block size=8.
Maximum @0.54V,25MHz ,3bit FFT, 1bit MAC,
block size=128, on AlexNet FC layer 2.

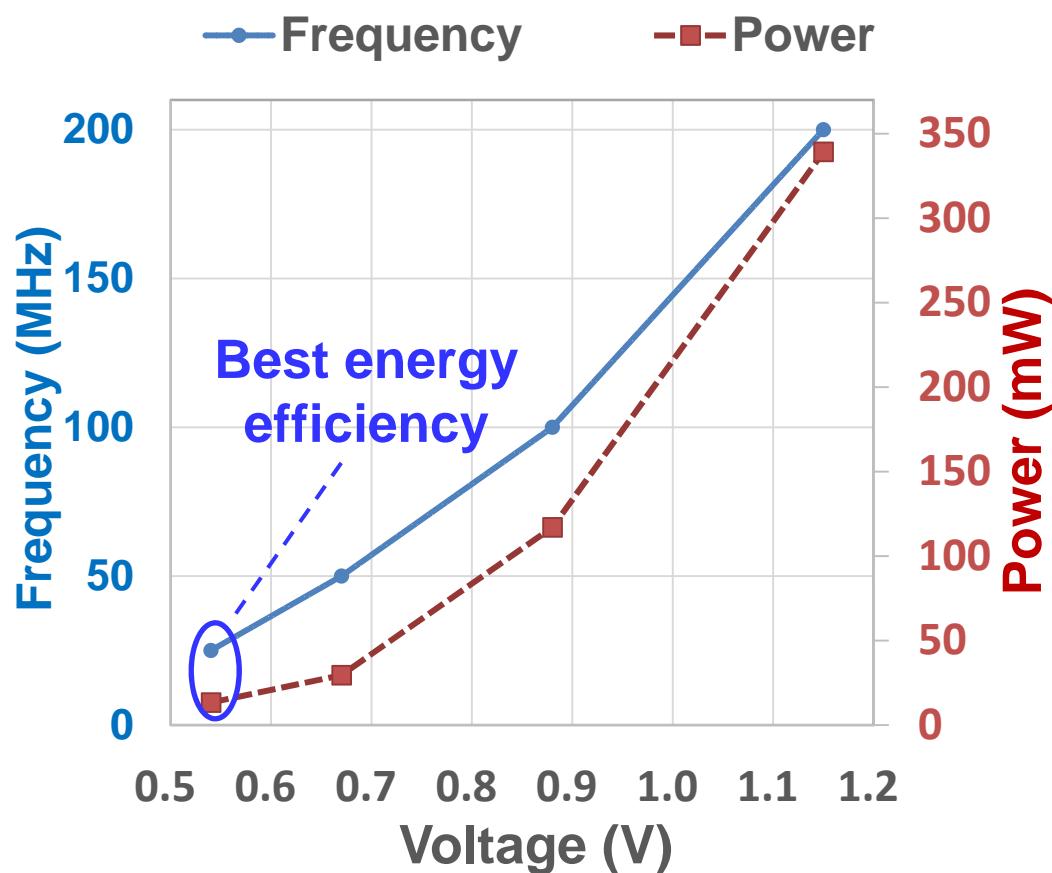
Chip Test Platform



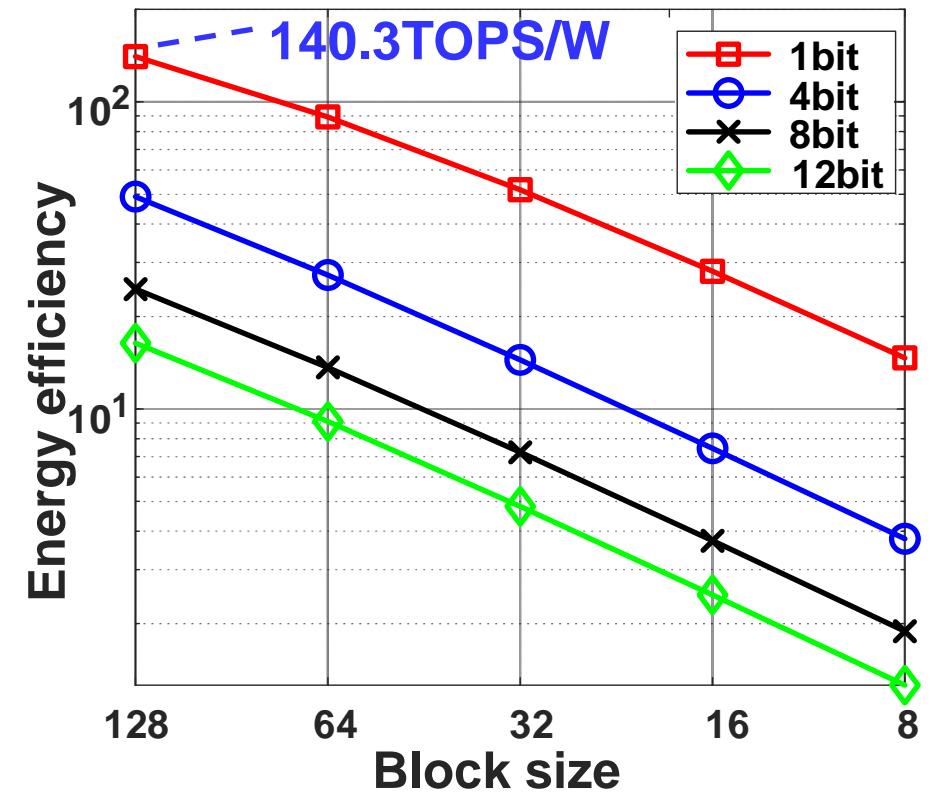
7.5: A 65nm 0.39-to-140.3TOPS/W 1-to-12b Unified Neural-Network Processor Using Block-Circulant-Enabled Transpose-Domain Acceleration with 8.1X Higher TOPS/mm² and 6T HBST-TRAM-Based 2D Data-Reuse Architecture

Performance Scaling

Voltage-frequency scaling



Block-size/bit-precision scaling



AlexNet FC layer 2 @0.54V,
25MHz, 3bit FFT, 1bit MAC.

Evaluation on Block-Circulant NN Models

Network	Tiny YOLO	LSTM	FC
Dataset	DataDJI	TIMIT	MNIST
Layer type	CNN	RNN (LSTM)	FC
Block-size	32	32	128
Accuracy loss	3%	1.93%	1.50%
# of circulant weights	494K	101K	17.4K
Equivalent operations	6.95GOP	6.50MOP	1.11MOP
FFT bit-precision	8bit	8bit	6bit
MAC bit-precision	5bit	4bit	2bit
Execution time (1 in 32 batch)	0.14s	33.8us	6.44us
Energy efficiency	3.76TOPS/W	14.4TOPS/W	12.9TOPS/W

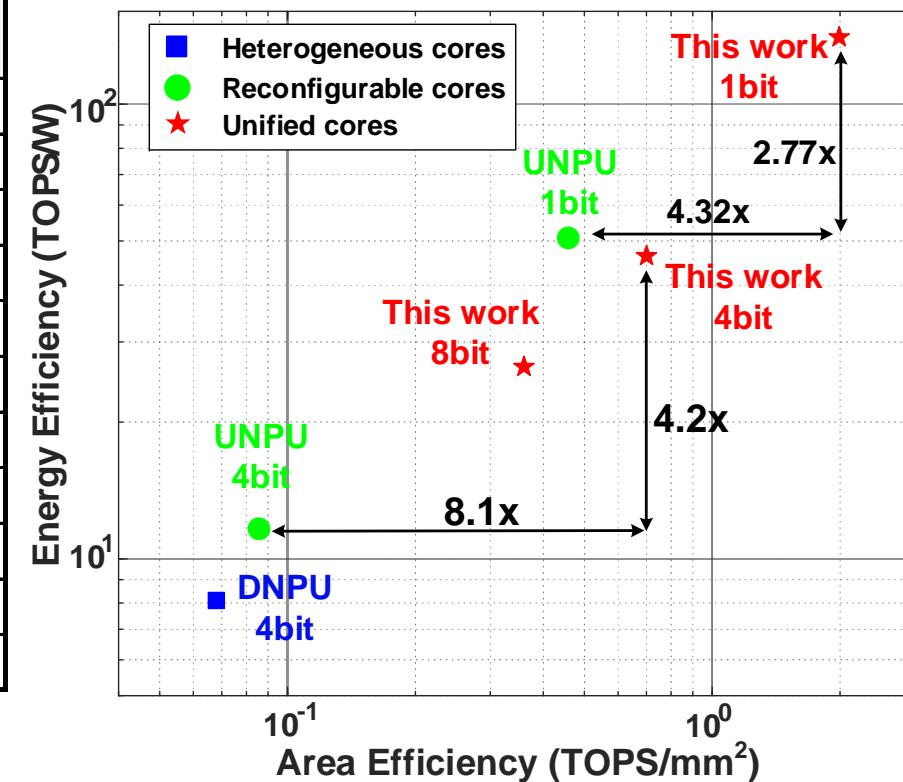
Performance Comparison

➤ Comparison with state-of-the-art NN processors

	ISSCC17-DNPU[2]	ISSCC18-UNPU[3]	VLSI18-STICKER[1]	This work
Technology	65nm	65nm LP	65nm GP	65nm GP
Area[mm ²]	16	16	12	7.5
Support NN type	CNN+FC+RNN	CNN+FC+RNN	CNN+FC	CNN+FC+RNN
Weight precision	4-7b(Quantized)	1-16b	8b	1-12b
Activation precision	4,8,16b	16b	8b	1-12b
Performance (TOPS)	1.08	7.37	5.6	14.9¹⁾
Power(mW)	34.6-279	3.2-297	20.5-248.4	13.3-339
Area efficiency (TOPS/mm ²)	0.068(4b) 0.79x	0.086(4b) 1x	0.47(Conv)	0.70(4b) 8.1x¹⁾
	-	0.46(1b) 1x	-	1.99(1b) 4.32x¹⁾
Energy efficiency (TOPS/W)	8.1(4b) 0.70x	11.6(4b) 1x	62.1 (Conv)	49.1(4b) 4.2x²⁾
	-	50.6(1b) 1x	-	140.3(1b) 2.77x²⁾

One operation (OP) represents one real multiplication or one real addition.

1) @1.15V,200MHz; 2) @0.54V,25MHz. AlexNet FC layer 2, block size 128.



Outline

- ❑ Introduction
- ❑ Challenges of Unified Block-Circulant Processor
- ❑ Proposed Unified NN Processor
 - Global-Parallel Bit-Serial FFT Module
 - 2-D Data-Reuse Array
 - HBST TRAM
- ❑ Measurement Results
- ❑ Conclusion

Conclusion

- ❑ STICKER-T : a transpose-domain accelerated NN processor

- Unified NN processing

- Supports CNN/FC/RNN in the same workflow**

- Global-parallel bit-serial FFT module

- Supports power-/area-efficient FFT operations**

- HBST TRAM and 2D-reuse MAC array

- Support efficient data transformation and data reuse**

A unified 1-to-12b NN processor with 8-128× storage reduction and 0.39-140.3TOPS/W energy efficiency

A 65nm 236.5nJ/Classification Neuromorphic Processor with 7.5% Energy Overhead On-Chip Learning Using Direct Spike-Only Feedback

Jeongwoo Park, Juyun Lee, and Dongsuk Jeon
Graduate School of Convergence Science and Technology
Seoul National University

Outline

I. Motivation

II. Neuromorphic Network

- I. General Overview

- II. Baseline Algorithm

- III. Algorithm Modification

III. Hardware Implementation

- I. Overall Architecture

- II. Out-of-order Weight Updates

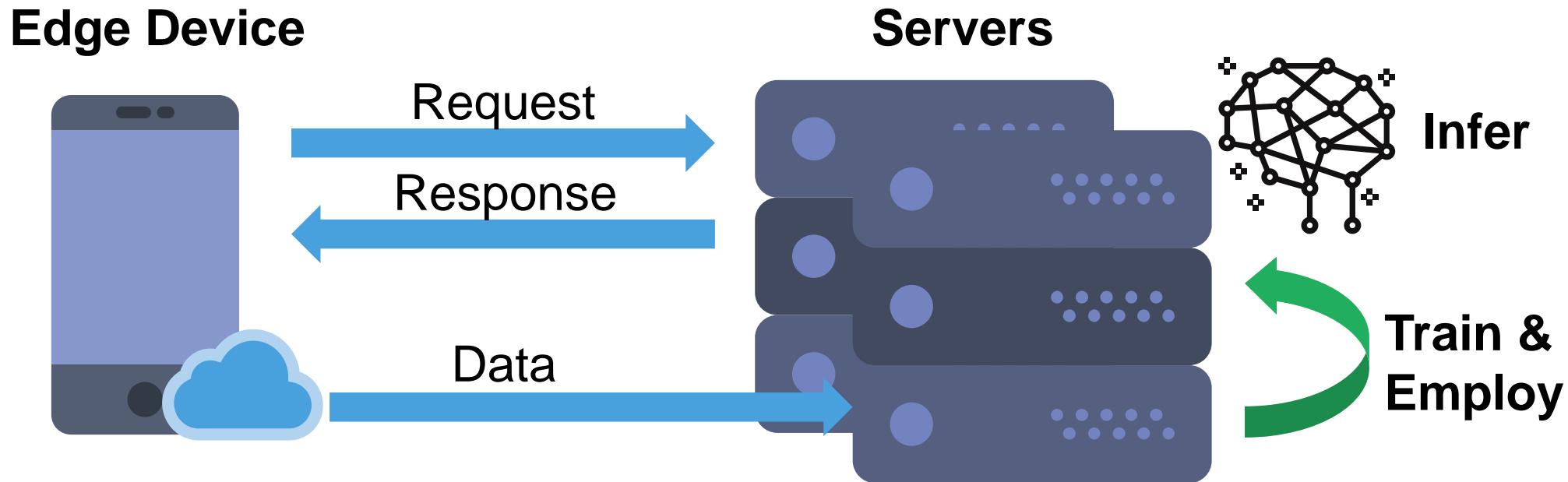
- III. Update Skipping Mechanism

IV. Measurements

V. Conclusion

Machine Learning Processing Schemes

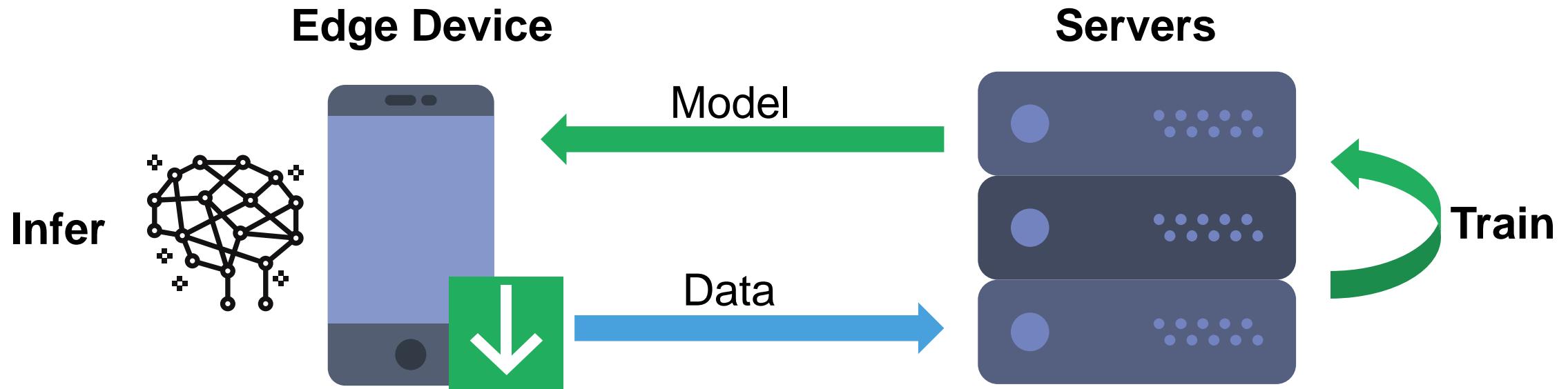
- Training in server, inference in server



- Run huge, powerful models
- Could only be run with network connection

Machine Learning Processing Schemes

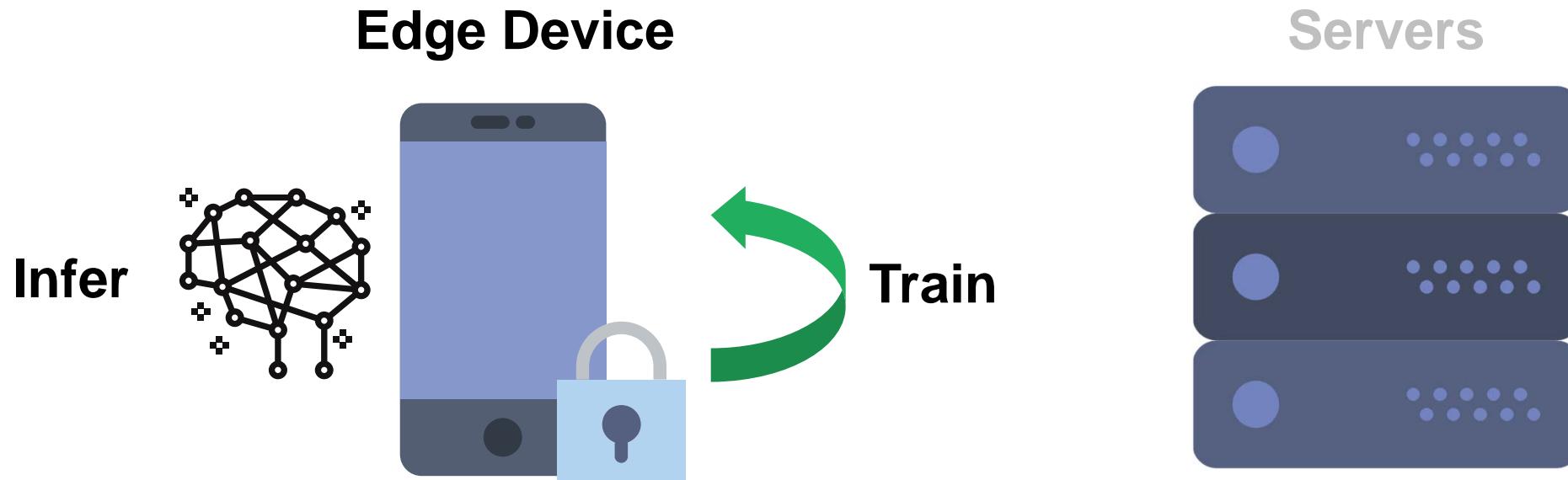
- Training in server, inference in edge device



- Run compact, efficient models
- Could be run offline once model is downloaded

Machine Learning Processing Schemes

- Training in edge device, inference in edge device

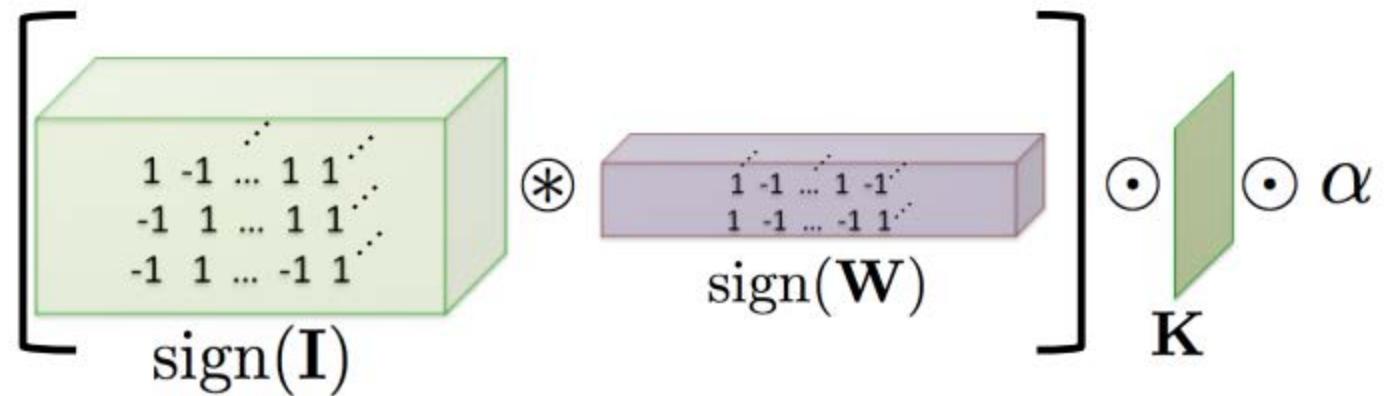


- Run compact, efficient models
- Adjusted offline reliably
- Protect private data

Training with Back Propagation

- Error tolerant inference operations
- Often **<8b** for inference

Convolution with XNOR-bitcounts^[1]

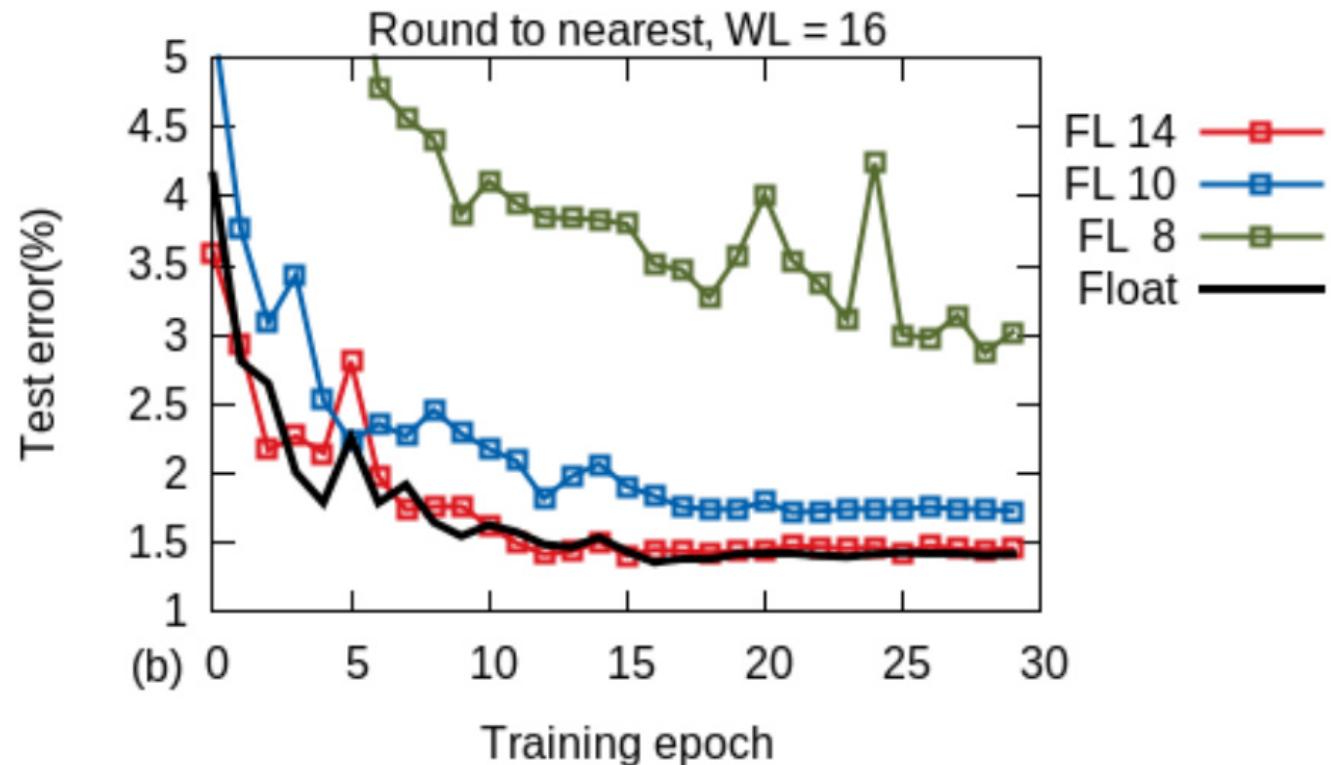


[1] Mohammad, ECCV '16.

Training with Back Propagation

- Error tolerant inference operations
- Often **<8b** for inference
- Training requires higher numerical precision
 - High-precision for error signals (propagating gradient)
 - Larger arithmetic units, more data transmission

MNIST Training With Fixed Point Numbers [2]



[2] Gupta, ICML '15.

Outline

I. Motivation

II. Neuromorphic Network

I. General Overview

II. Algorithm Modification

III. Hardware Implementation

I. Overall Architecture

II. Out-of-order Weight Updates

III. Update Skipping Mechanism

IV. Measurements

V. Conclusion

Neuromorphic Networks

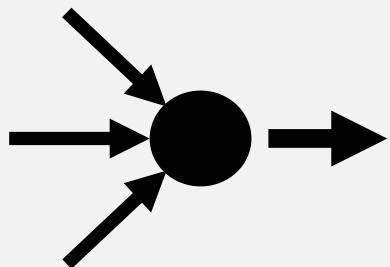
- Neuromorphic networks are neural networks that are bio-plausible and brain-imitating
- Bio-plausibility in
 - Neuron Models
 - Learning Rules

Neuromorphic Neuron Models

- Bio-plausible neuron models

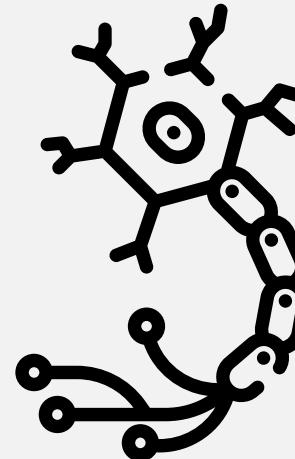
Conventional Neuron Model

Point Neurons



- Real number input / output
- Simple multiply-accumulate operations

Bio-Plausible Neuron Models

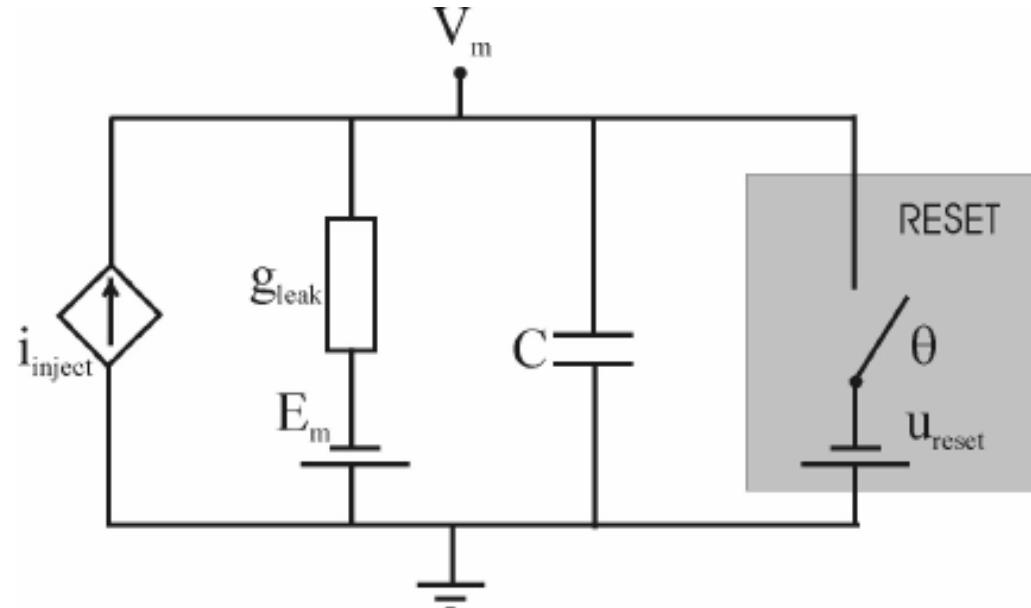


- Spike (1-bit) input / output
- Differential equation-based neuron behavior
- Time domain

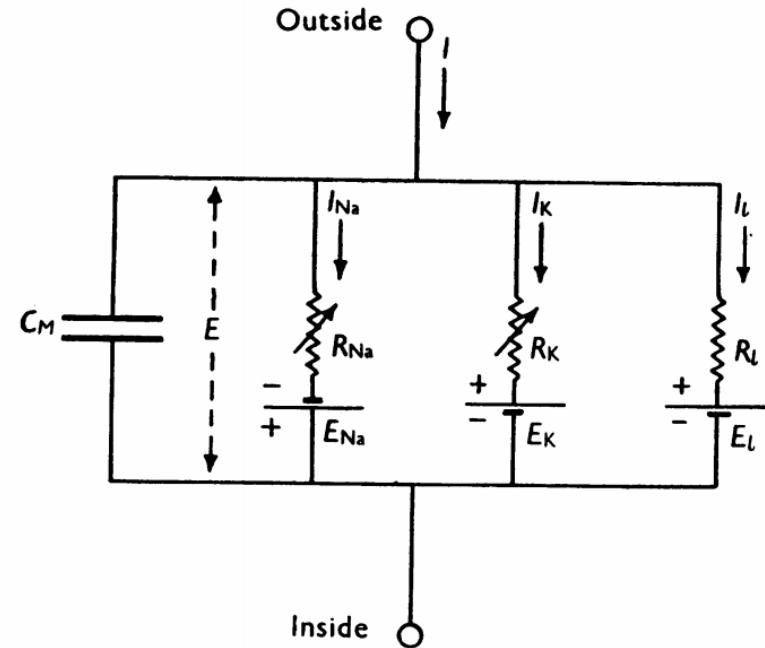
Bio-Plausible Neuron Model Examples

- Bio-plausible neuron models

Leaky Integrate-and-Fire Model [3]



Hodgkin-Huxley Model [4]



[3] Kraft, WODES '06

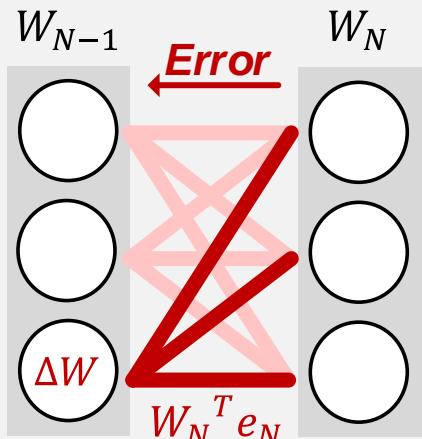
[4] Hodgkin, J. Physiol. '52

Neuromorphic Learning Rules

- Bio-plausible learning rules

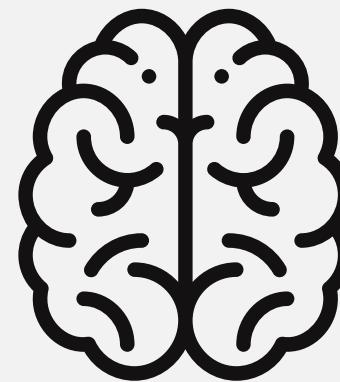
Conventional Learning Rule

Back Propagation



- High accuracy & performance
- High precision error signals
- Global error propagation

Bio-Plausible Learning Rules

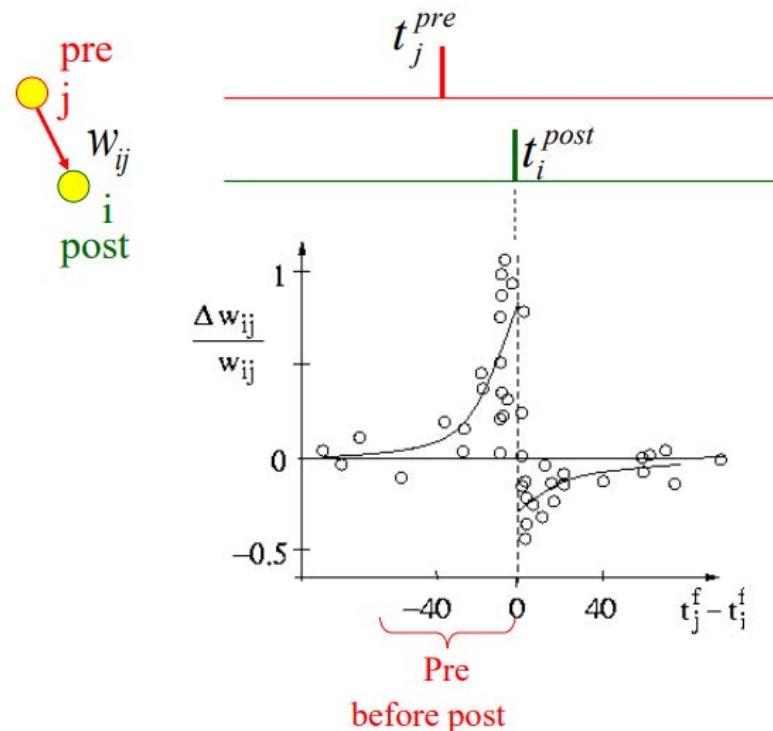


- Received spikes for a neuron as error signals
- Energy efficient
- Lower accuracy

Bio-Plausible Learning Rule Examples

- Bio-plausible learning rules

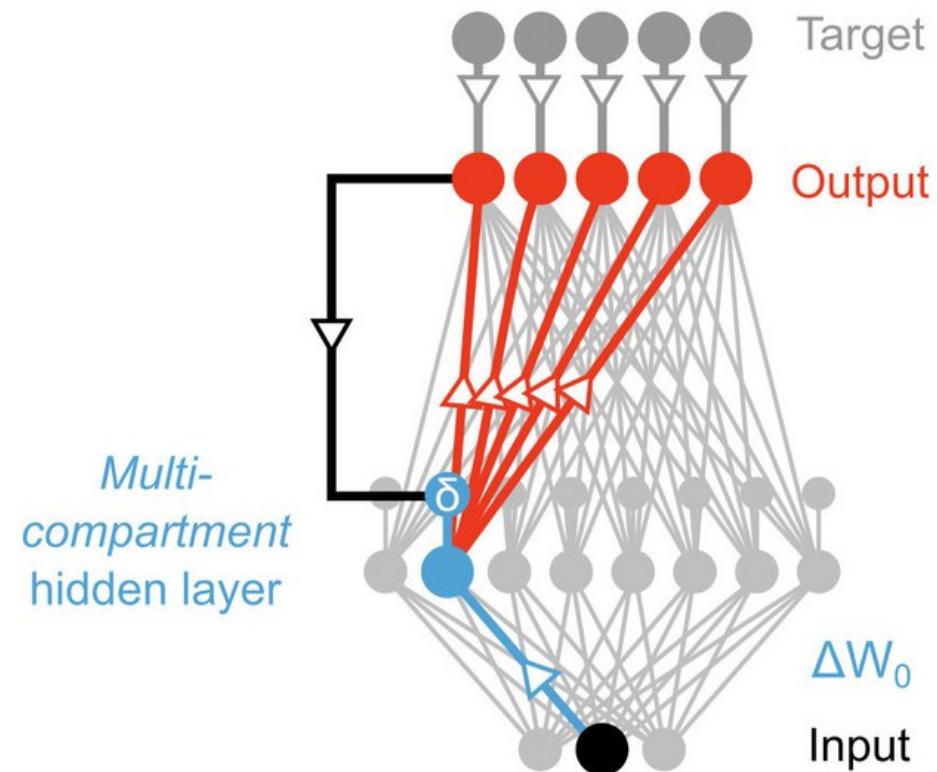
Spike Timing Dependent Plasticity [5]



[5] Sjostrom, STDP '10

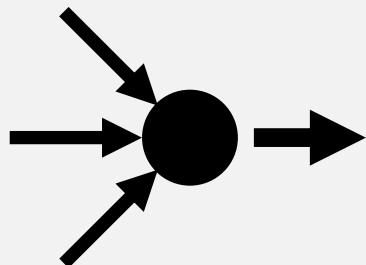
[6] Guerguiev, Elife '17

Segregated Dendrites [6]



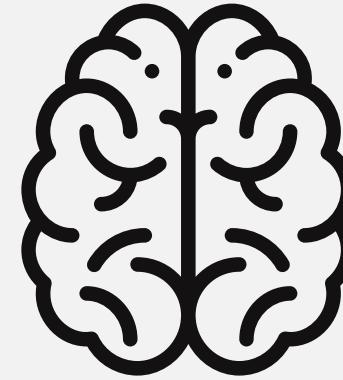
Bringing the Best of Two Worlds Together

Proposed Neuron Model



- Spike (1-bit) input / output
- Simple accumulate operations

Proposed Learning Rule



- Received spikes for a neuron as error signals
- Energy efficient
- High accuracy

- Good energy efficiency compared to inference-only machines

- Low training energy overhead
- Classification accuracy comparable to back-propagation

Outline

I. Motivation

II. Neuromorphic Network

I. General Overview

II. Algorithm Modification

III. Hardware Implementation

I. Overall Architecture

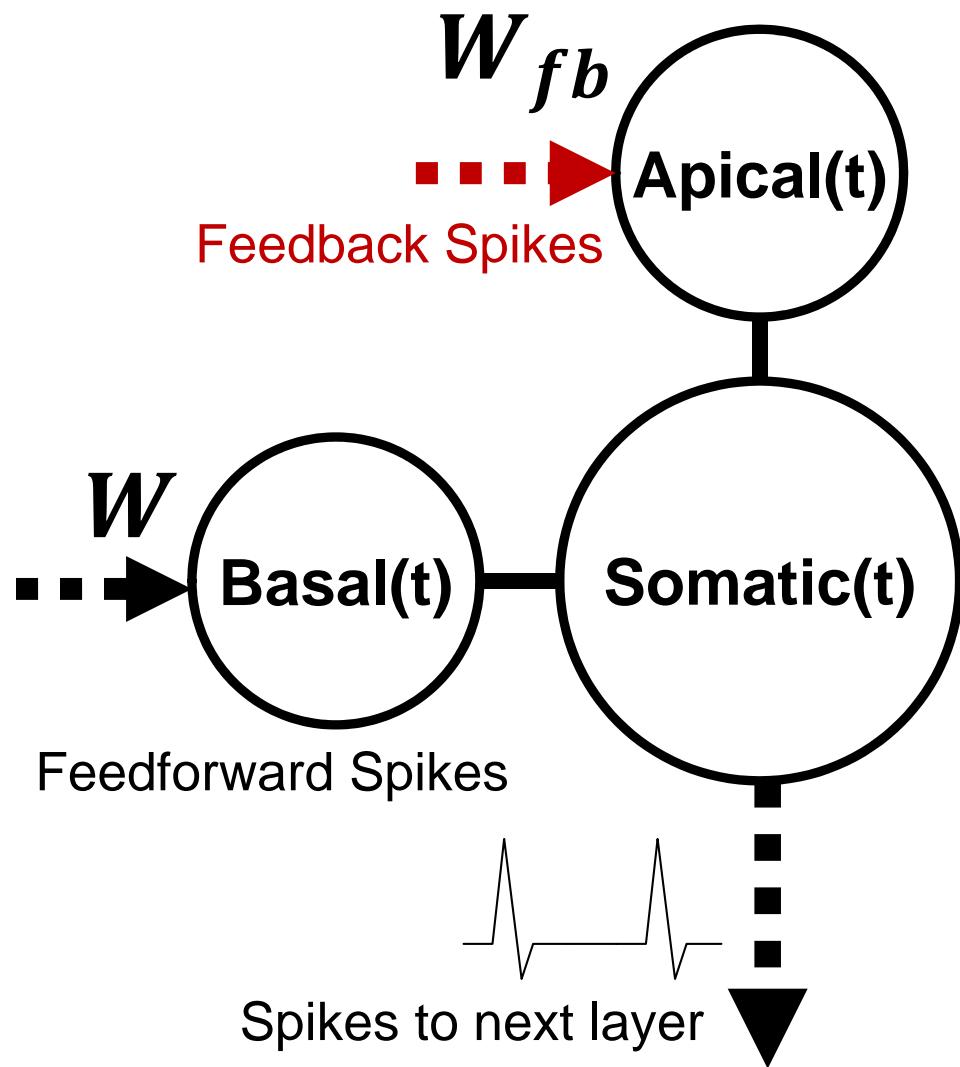
II. Out-of-order Weight Updates

III. Update Skipping Mechanism

IV. Measurements

V. Conclusion

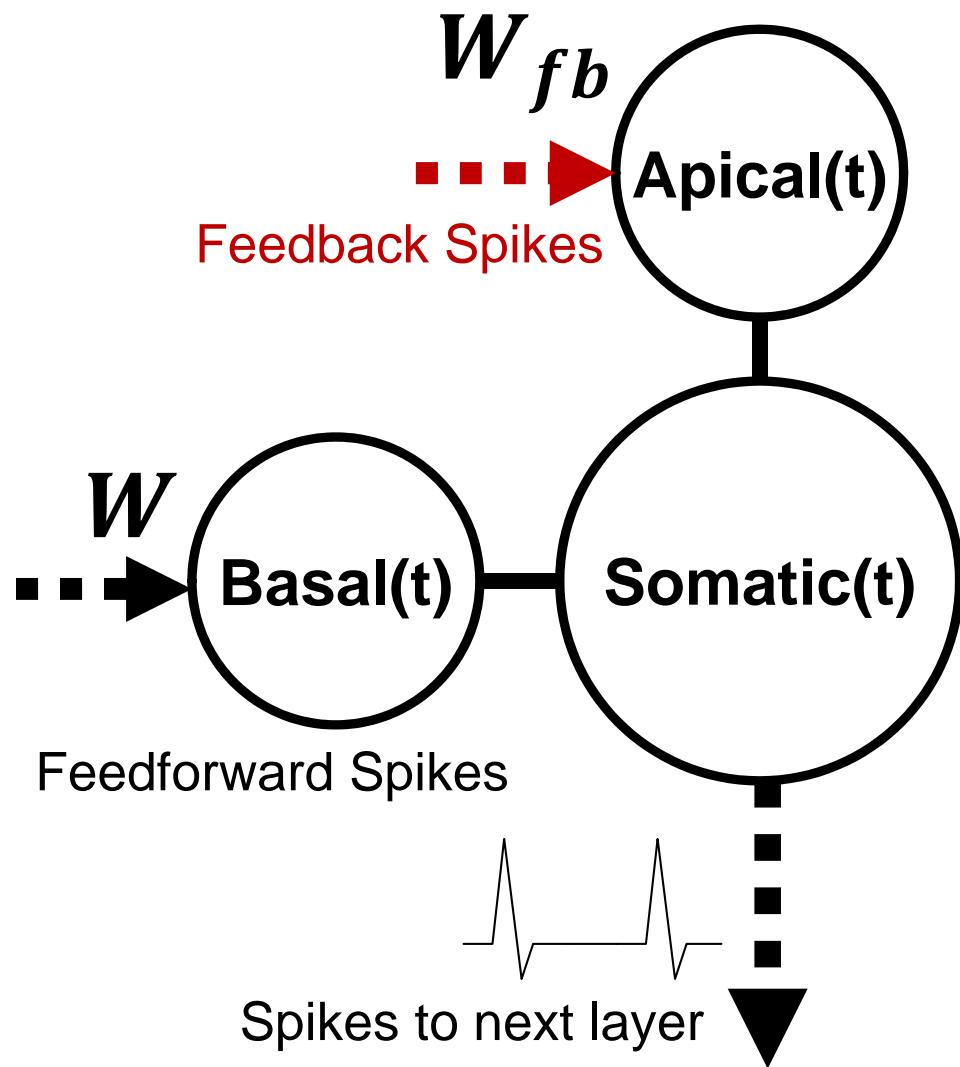
Neuron Model of Segregated Dendrites^[6]



- Internal neuron dynamics mainly consist of three dendrite values:
 - Apical (A) – Feedback receiver
 - Basal (B) – Feedforward receiver
 - Somatic (S) – Determines spike

[6] Guerguiev, Elife '17

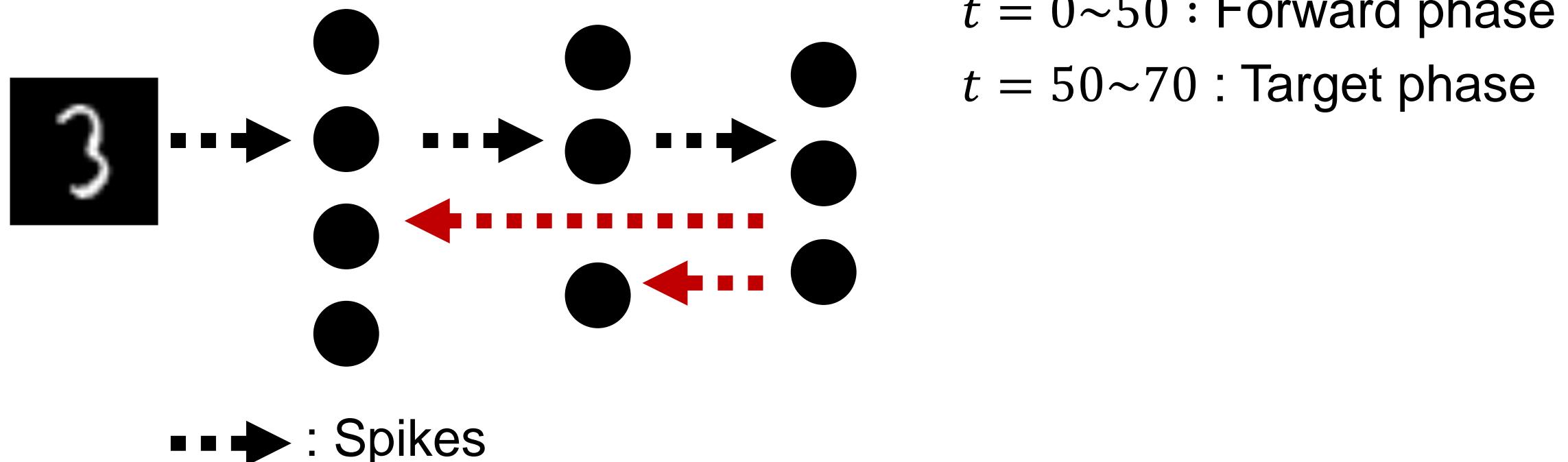
Neuron Model of Segregated Dendrites



- Internal neuron dynamics mainly consist of three dendrite values:
 - Apical (A) – Feedback receiver
 - Basal (B) – Feedforward receiver
 - Somatic (S) – Determines spike
- A and B affect S through time domain:**
$$\frac{dS}{dt} = g_L S + g_B(B - S) + g_A(A - S)$$
- Spikes are fired if $\sigma(S) > \text{threshold}$

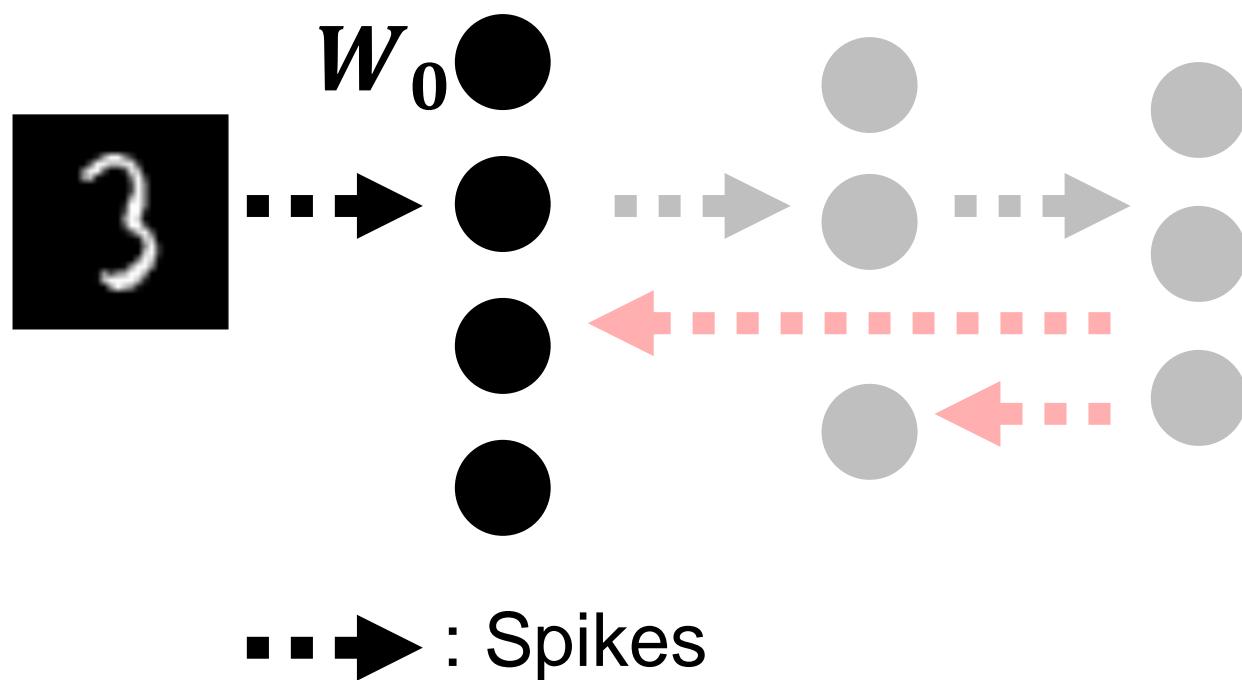
Learning Rule of Segregated Dendrites

- Feedforward structure, supervised learning
- Neuron model with dynamic differential equation: time domain

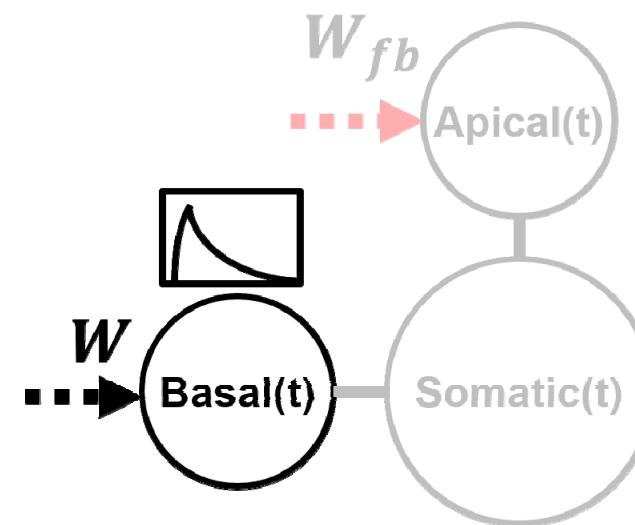


Learning Rule of Segregated Dendrites

- Feedforward structure, supervised learning
- Neuron model with dynamic differential equation : time domain

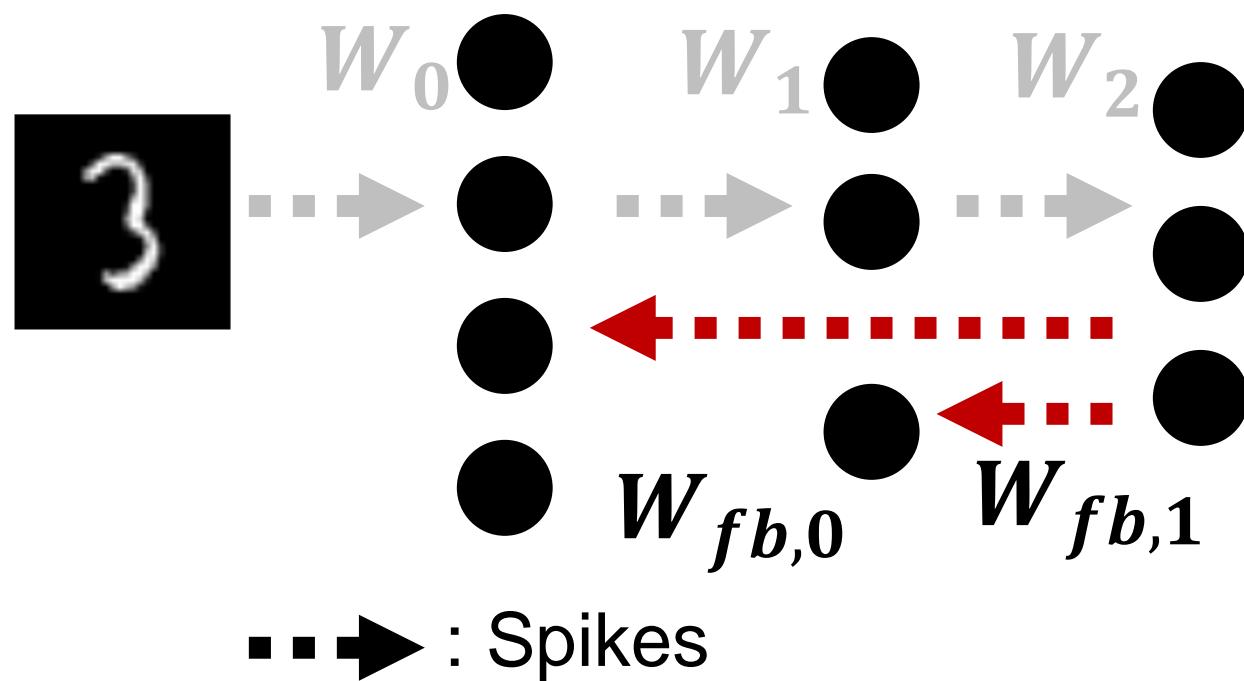


$t = 0 \sim 50$: Forward phase
 $t = 50 \sim 70$: Target phase

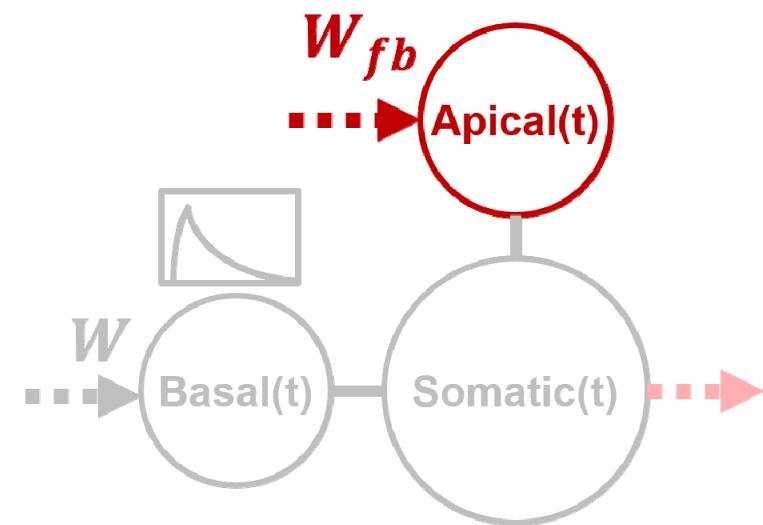


Learning Rule of Segregated Dendrites

- Feedforward structure, supervised learning
- Neuron model with dynamic differential equation : time domain

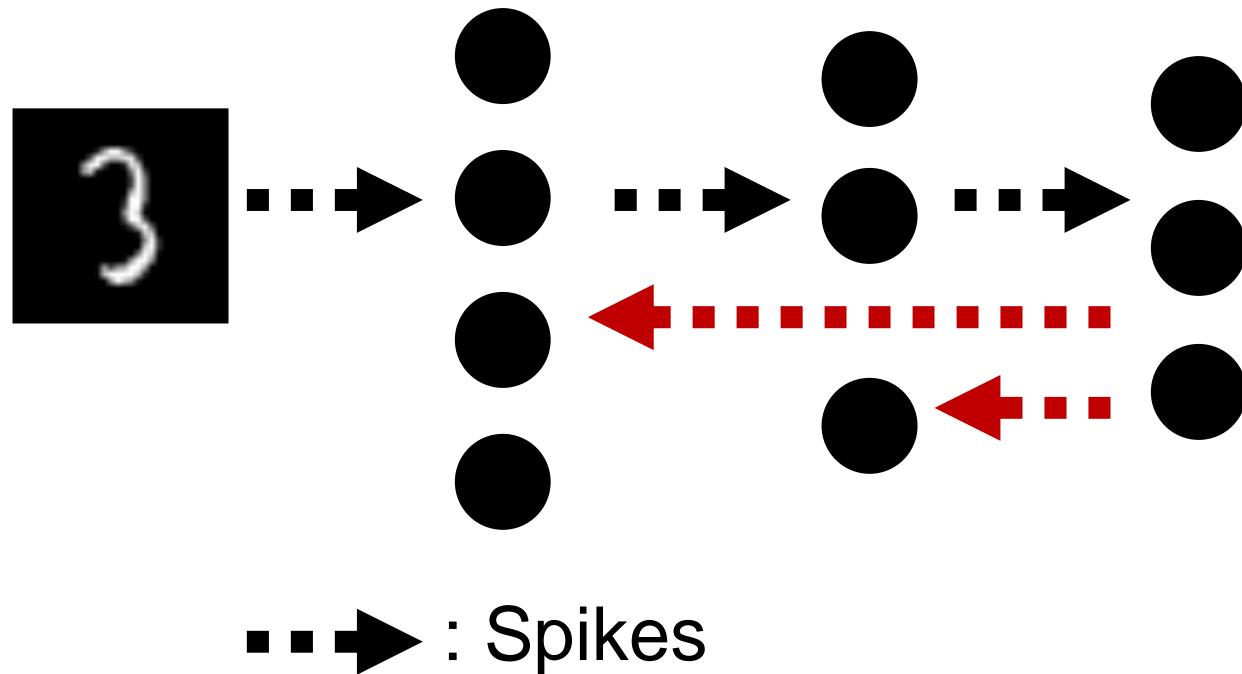


$t = 0 \sim 50$: Forward phase
 $t = 50 \sim 70$: Target phase



Learning Rule of Segregated Dendrites

- Feedforward structure, supervised learning
- Neuron model with dynamic differential equation : time domain



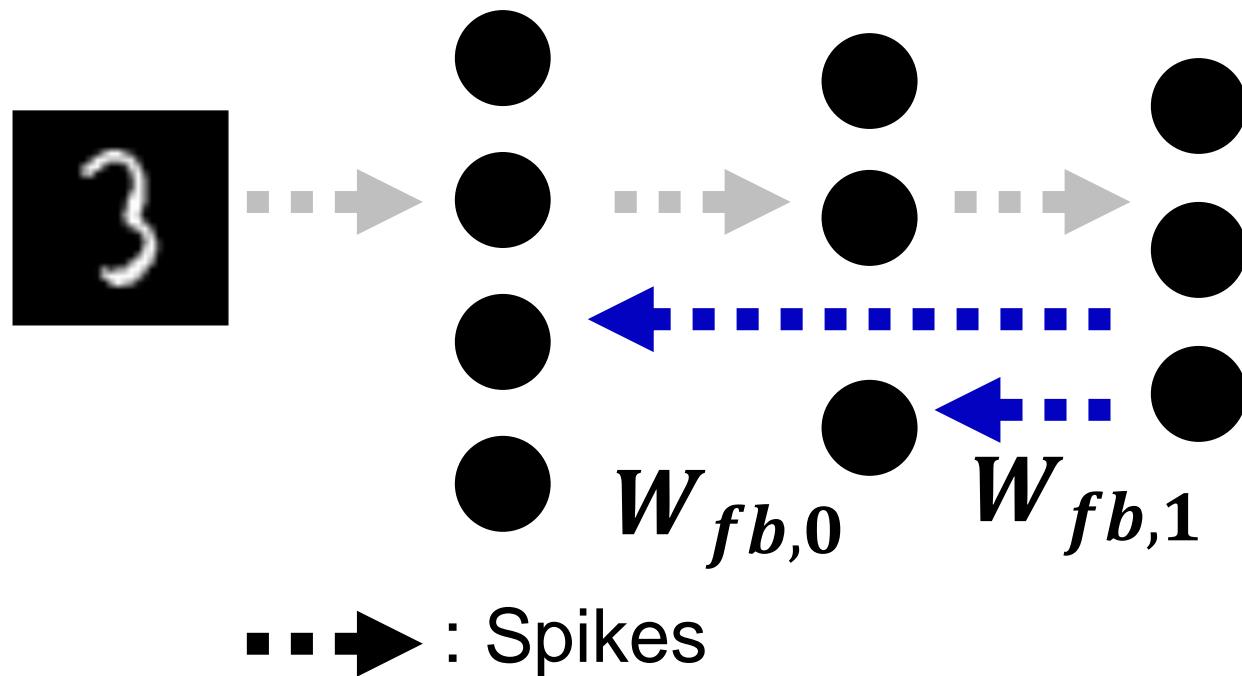
$t = 0 \sim 50$: Forward phase

$t = 50 \sim 70$: Target phase

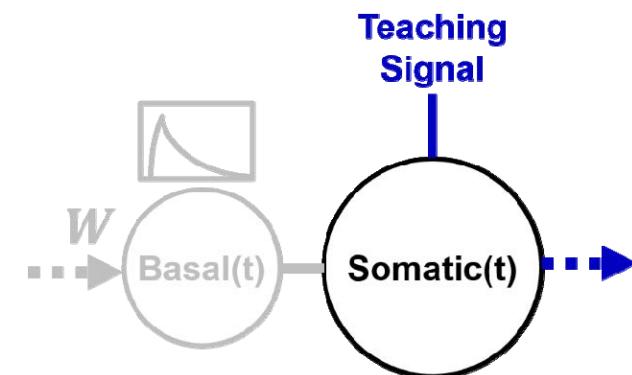
- This happens in each time step (0~50)
- After reaching “plateau”, average values of apical / somatic are saved

Learning Rule of Segregated Dendrites

- Feedforward structure, supervised learning
- Neuron model with dynamic differential equation : time domain

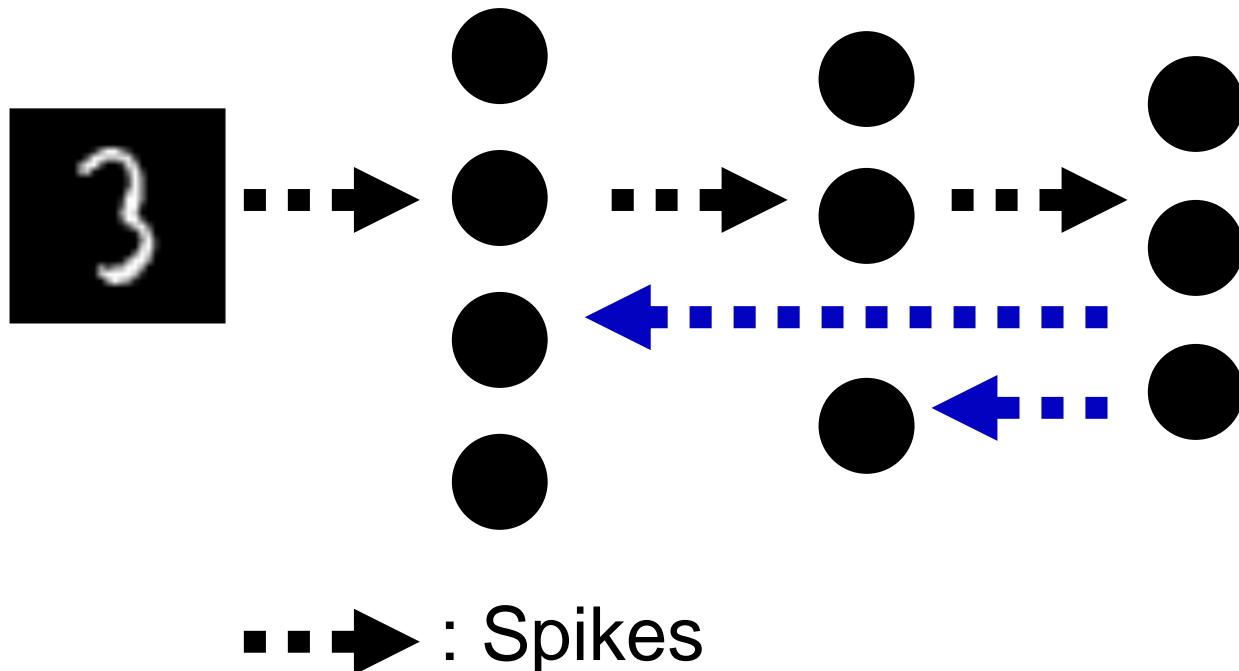


$t = 0 \sim 50$: Forward phase
 $t = 50 \sim 70$: Target phase



Learning Rule of Segregated Dendrites

- Feedforward structure, supervised learning
- Neuron model with dynamic differential equation : time domain



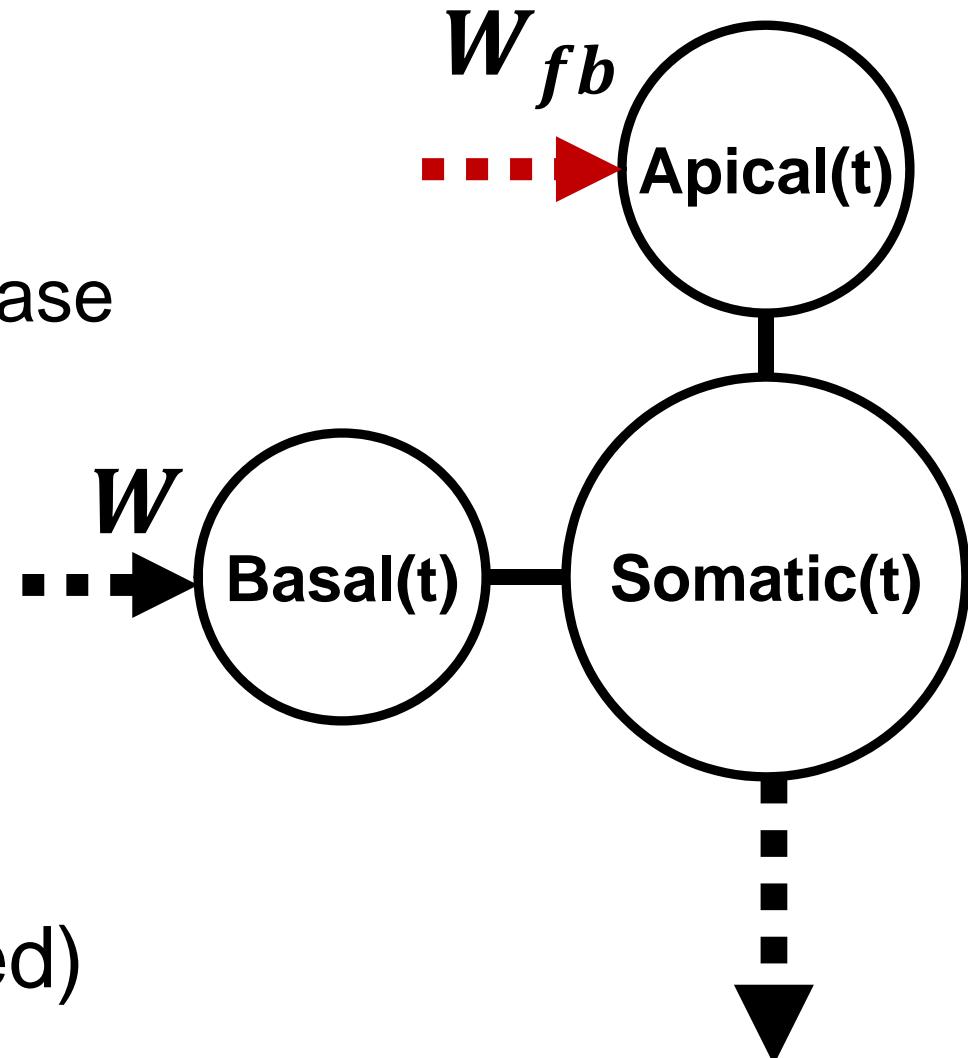
$t = 0 \sim 50$: Forward phase

$t = 50 \sim 70$: Target phase

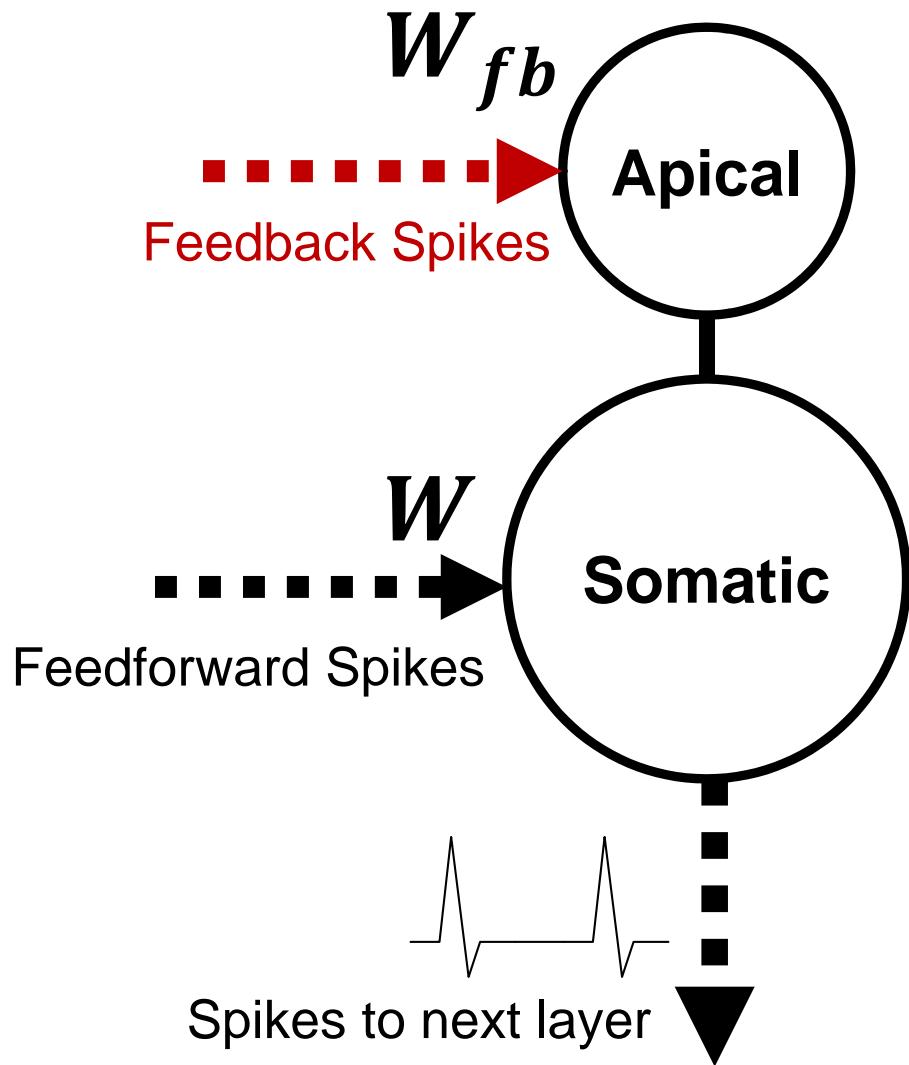
- This happens in each time step (50~70)
- After reaching plateau, average values of **apical** are saved

Learning Rule of Segregated Dendrites

- Saved Values:
 - Somatic of forward phase (s)
 - Apical of forward(a_f), target (a_t) phase
- Calculating ΔW
 - $\Delta FiringProbability = \sigma'(s)$
 - $\Delta Dendrites = \sigma(a_t) - \sigma(a_o)$
 - $\Delta W = \Delta FireProb * \Delta Dendrites$
- W_{fb} is not trained (random & fixed)

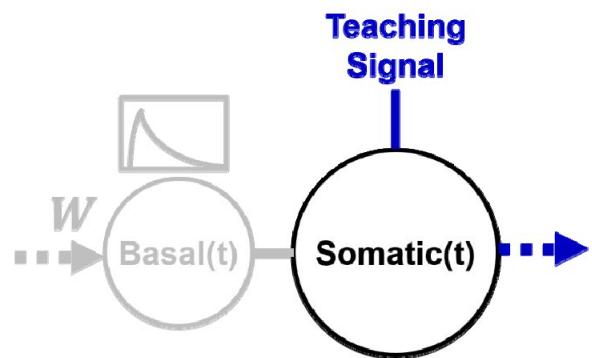
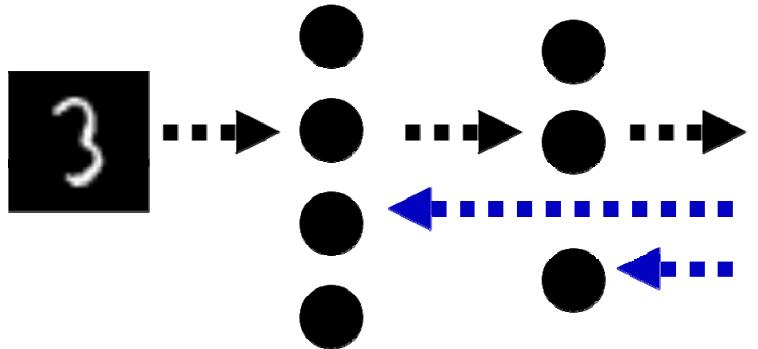


Modified Segregated Dendrites



- Simplified internal neuron dynamics
 - Apical (A) – Feedback receiver
 - Basal (B) – Feedforward receiver
 - Somatic (S) – Determines spike
- Somatic calculated simply by :
 - $\text{Somatic} = \sum W_i \cdot \text{spikes}_i$

Modified Segregated Dendrites



- Simplified internal neuron dynamics
 - Apical (A) – Feedback receiver
 - Basal (B) – Feedforward receiver
 - Somatic (S) – Determines spike
- Somatic calculated simply by :
 - $\text{Somatic} = \sum W_i \cdot \text{spikes}_i$
- Target spikes are simple one-hot encoding of labels

Single-shot Modified Segregated Dendrites

- NOT in time domain anymore

$$\frac{dS}{dt} = g_L S + g_B(B - S) + g_A(A - S) \rightarrow \text{Somatic} = \sum W_i \cdot \text{spikes}_i$$

- No need for “plateau” to be reached
- Single sample instead of 50 time steps
- Sampled values instead of average values

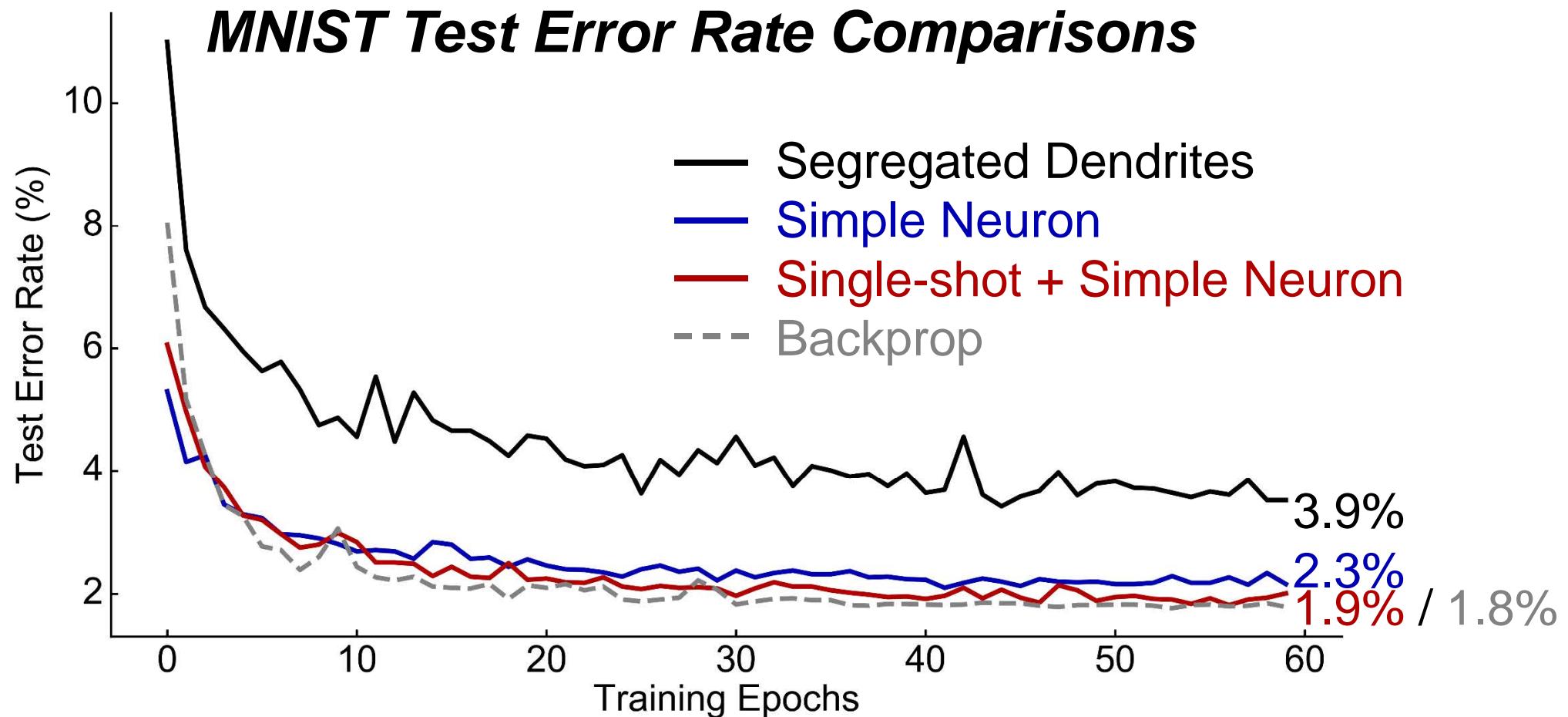
Modified Segregated Dendrites

	Clock Cycles /Image	Training Buffer Memory Size	OP/Image
Segregated Dendrites	14,000	47.2Kb	29,379KOP
Modified SD	10,010	38.4Kb	20,741KOP
Single-shot Modified SD	212	7.6Kb	804KOP
Total Improvement	x66.0	x6.2	x36.5

- Single-shot dramatically reduces computation
 - Reduce clock cycles by ×50
- Modified neuron architecture simplifies neuron computation

Boosted Classification Performance

- Both of these modifications improve classification performance



Outline

I. Motivation

II. Neuromorphic Network

I. General Overview

II. Algorithm Modification

III. Hardware Implementation

I. Overall Architecture

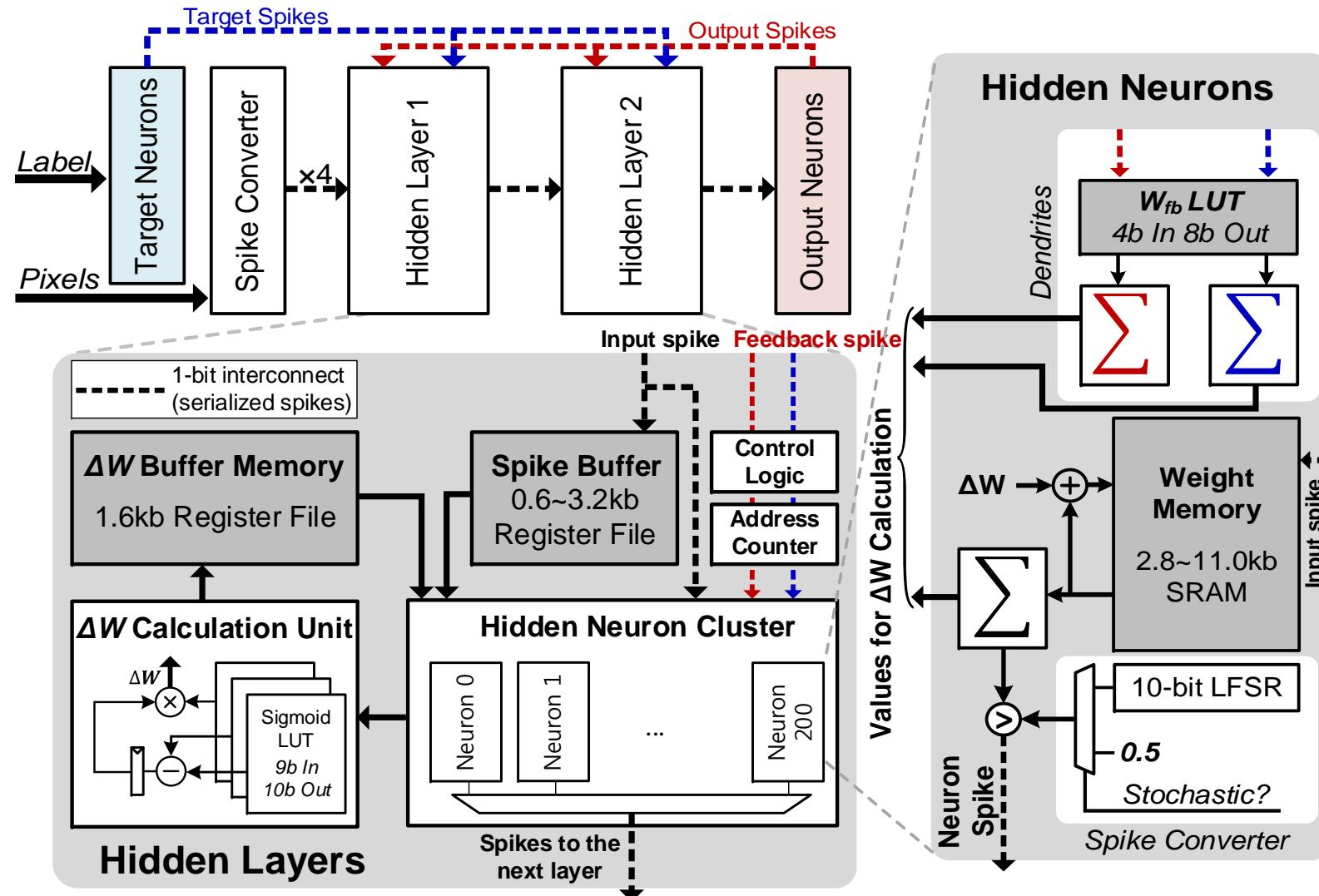
II. Out-of-order Weight Updates

III. Update Skipping Mechanism

IV. Measurements

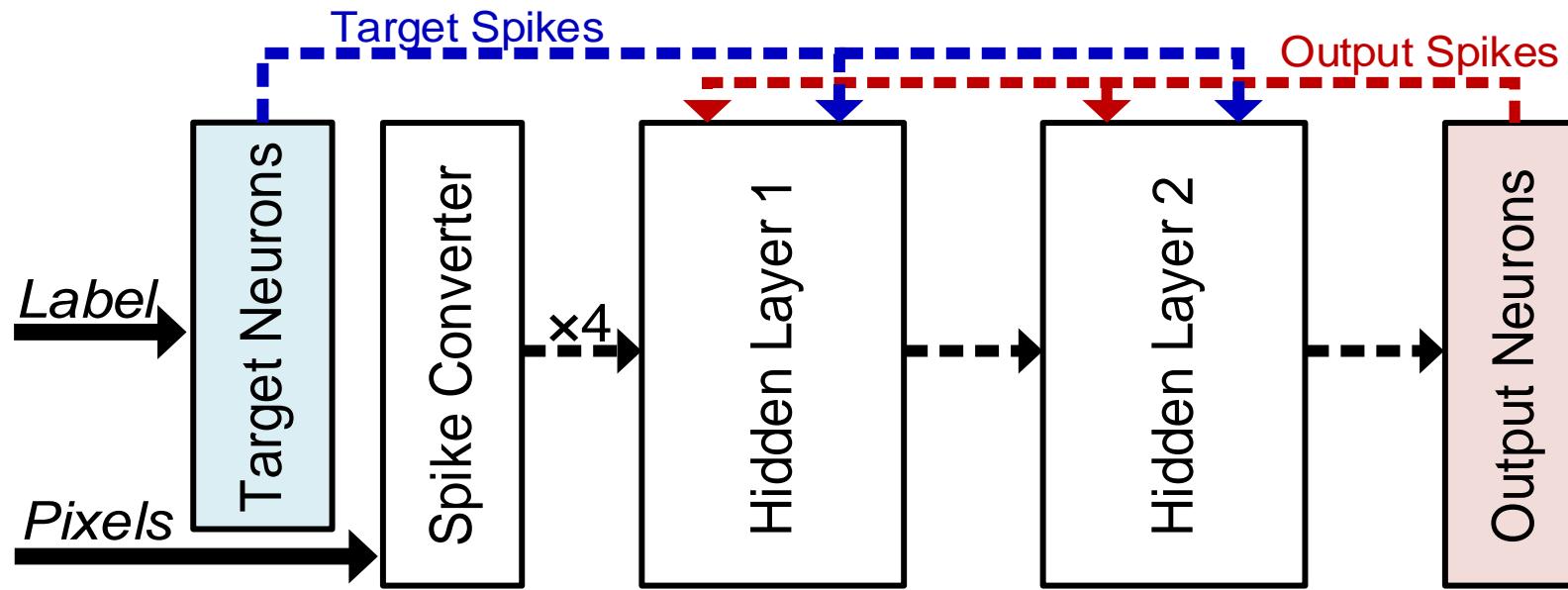
V. Conclusion

Overall Architecture of Proposed Design



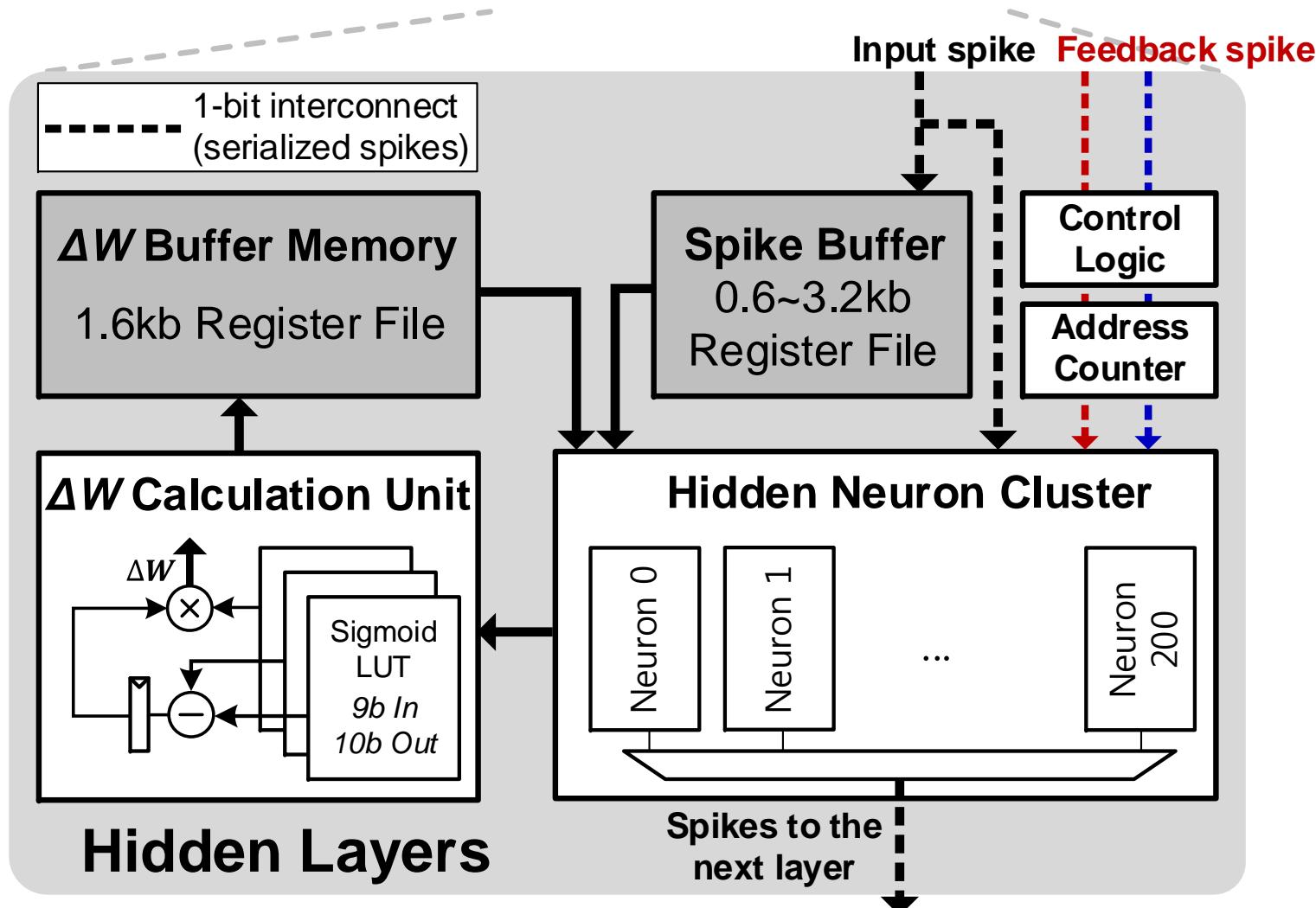
- 784-200-200-10 Architecture for MNIST classification
- Hardware-embedded digital neurons
- Serialized 1-bit interconnect between layers

Overall Architecture of Proposed Design



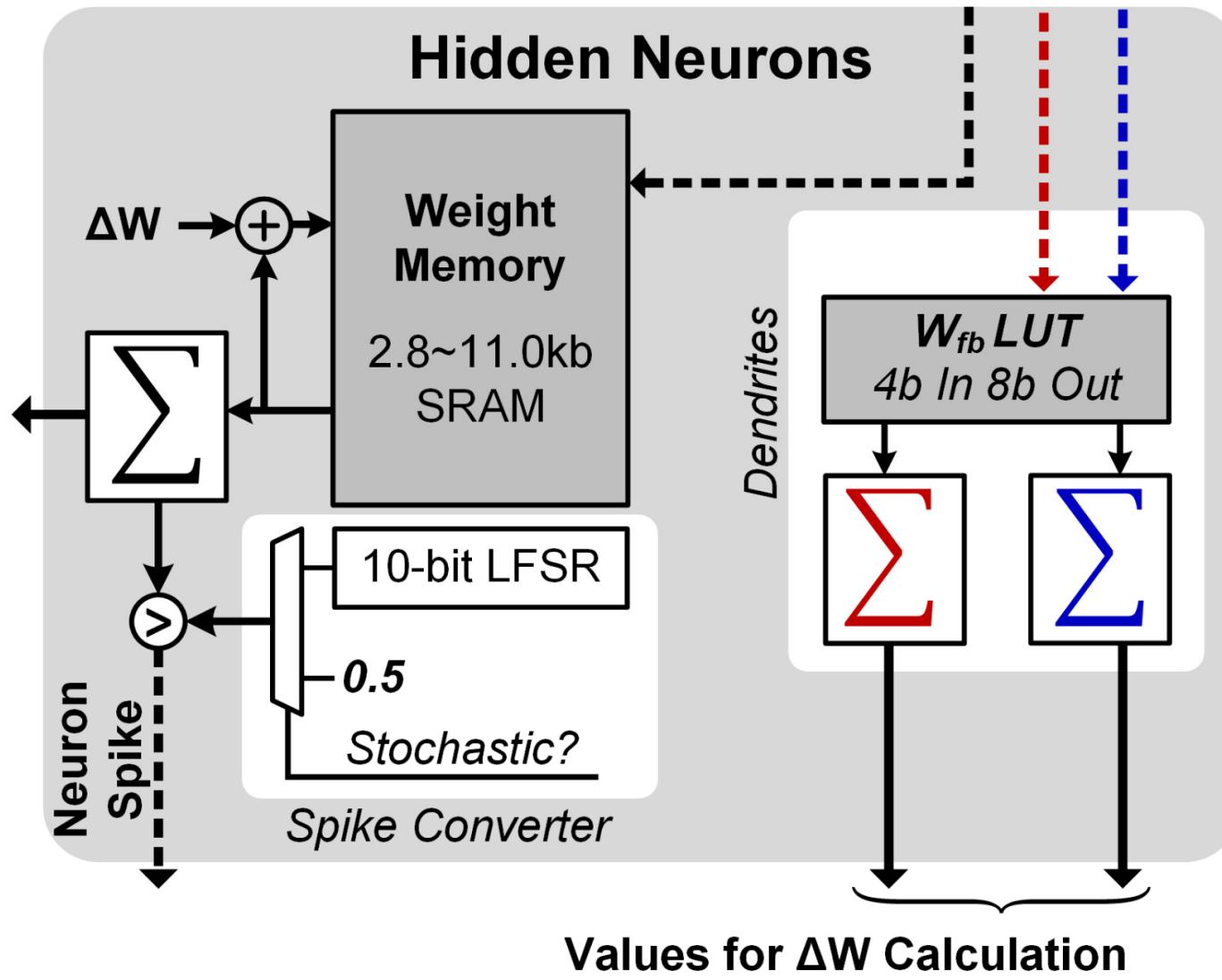
- 784-200-200-10 Architecture for MNIST classification
- Hardware-embedded digital neurons
- Serialized 1-bit interconnect between layers

Architecture of Neuron Layers



- Each layer sequentially accepts & sends 1-bit spikes
- Consists of:
 - Neuron cluster
 - Update value calculation unit
 - Sigmoid LUT
 - 2.2Kb / 4.8Kb Buffer memory

Architecture of Neurons



- Accumulate weight if input spike==1
- Random threshold determines spike
- Translate output / target spikes to apical dendrites
- 5 Neurons share SRAM for area reduction

Outline

I. Motivation

II. Neuromorphic Network

I. General Overview

II. Algorithm Modification

III. Hardware Implementation

I. Overall Architecture

II. Out-of-order Weight Updates

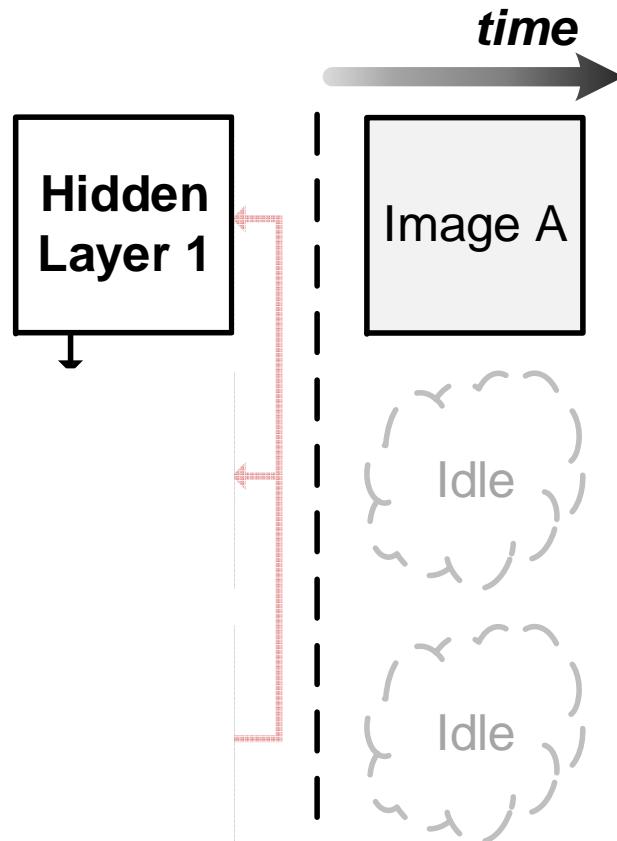
III. Update Skipping Mechanism

IV. Measurements

V. Conclusion

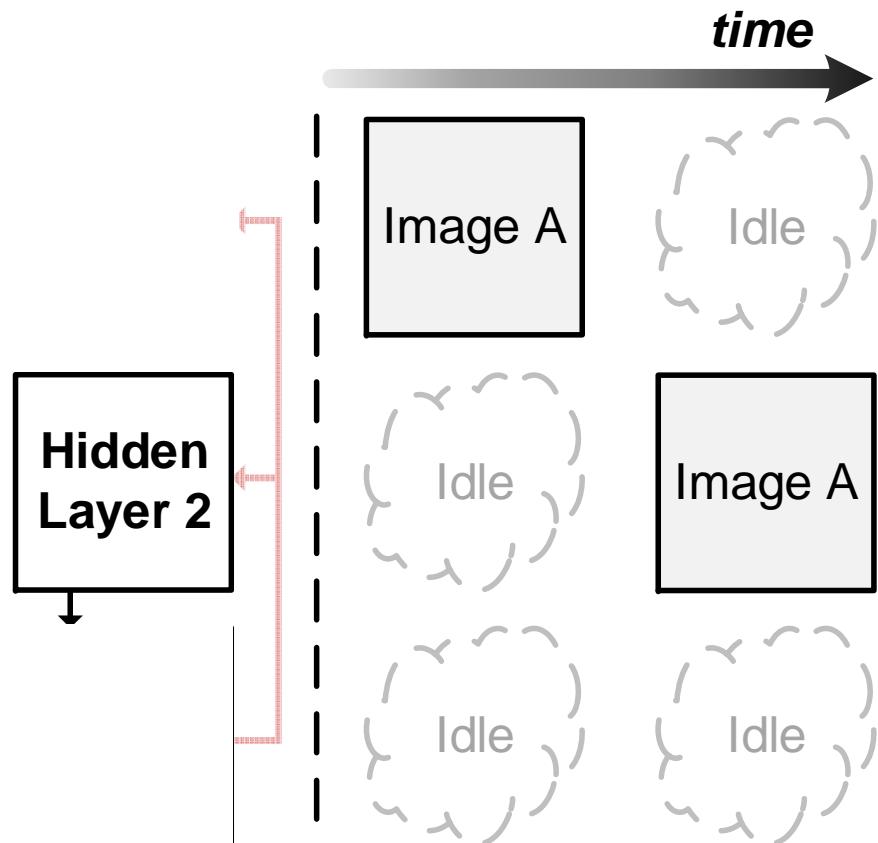
Conventional In-Order Updates

- ΔW values for an image updated **before** processing next image



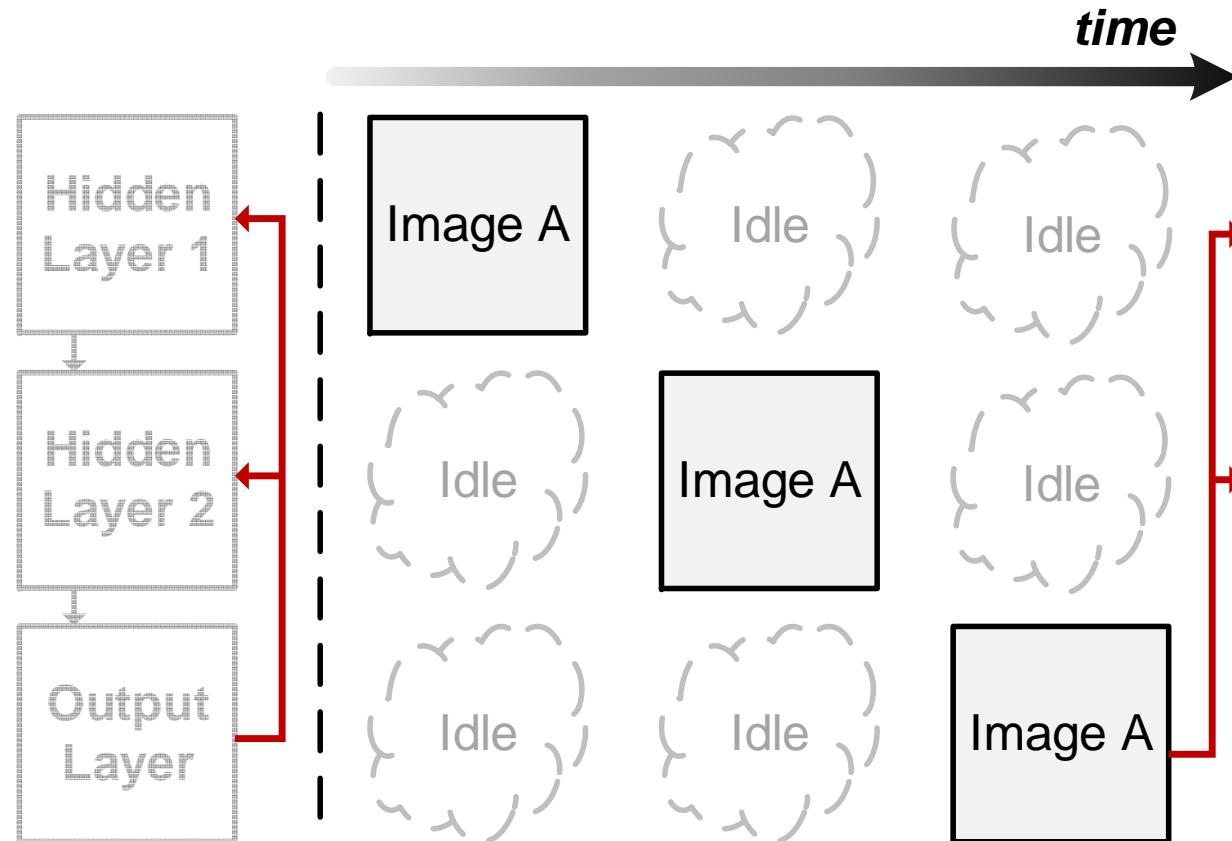
Conventional In-Order Updates

- ΔW values for an image updated **before** processing next image



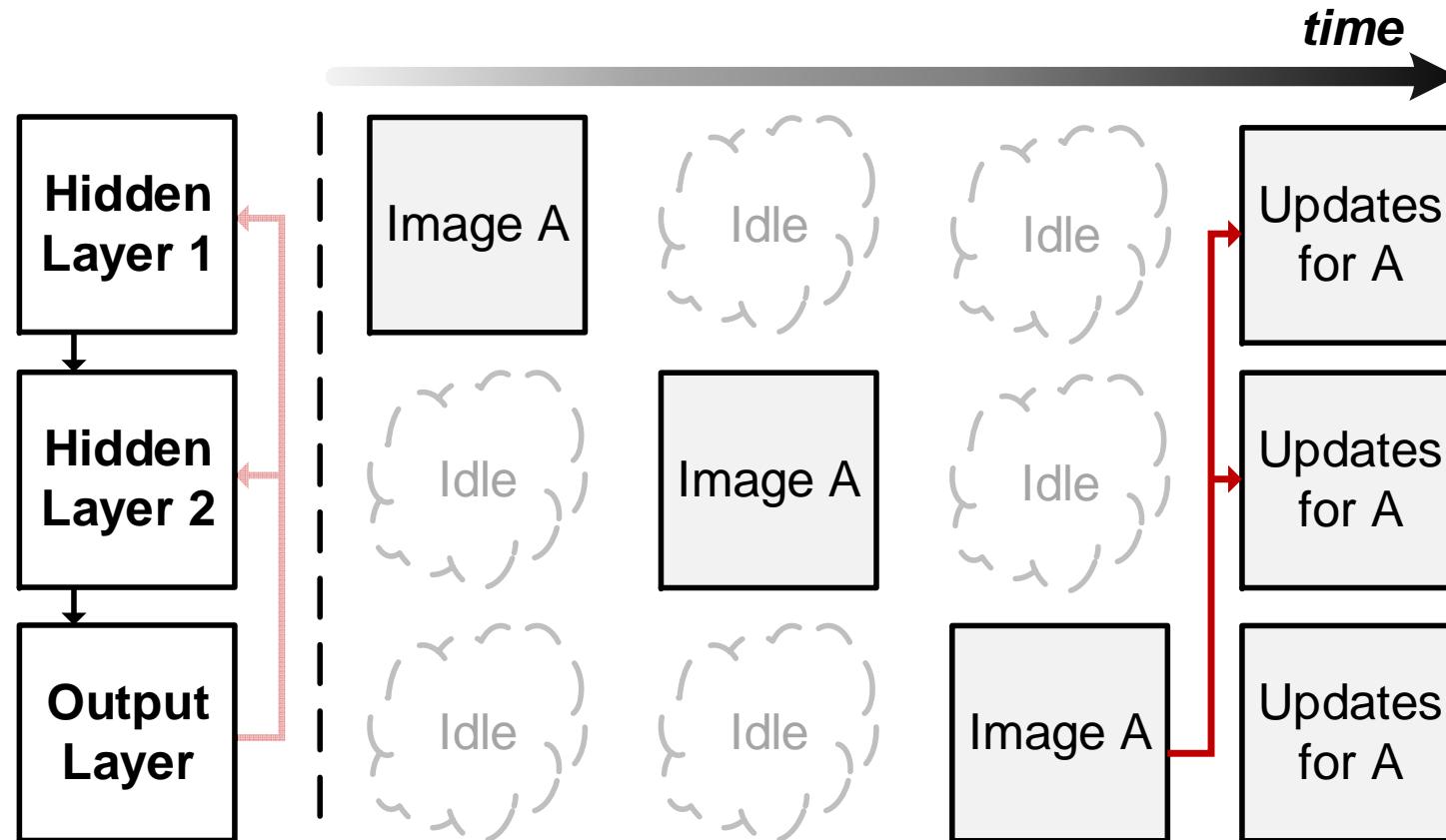
Conventional In-Order Updates

- ΔW values for an image updated **before** processing next image



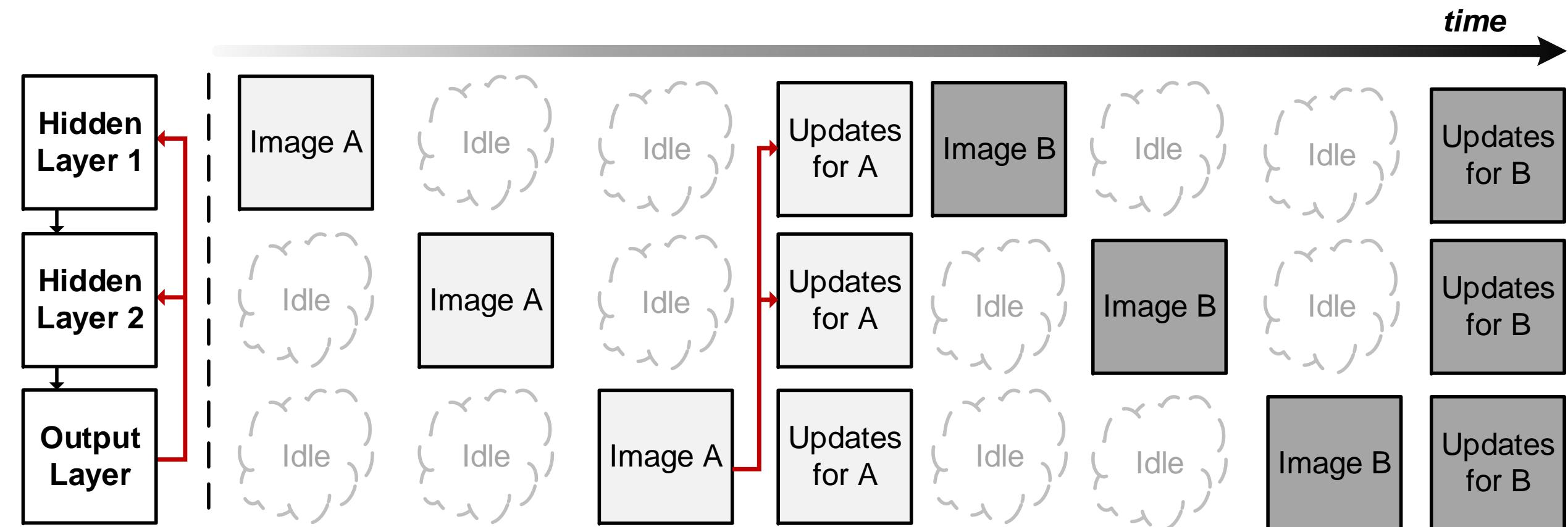
Conventional In-Order Updates

- ΔW values for an image updated **before** processing next image



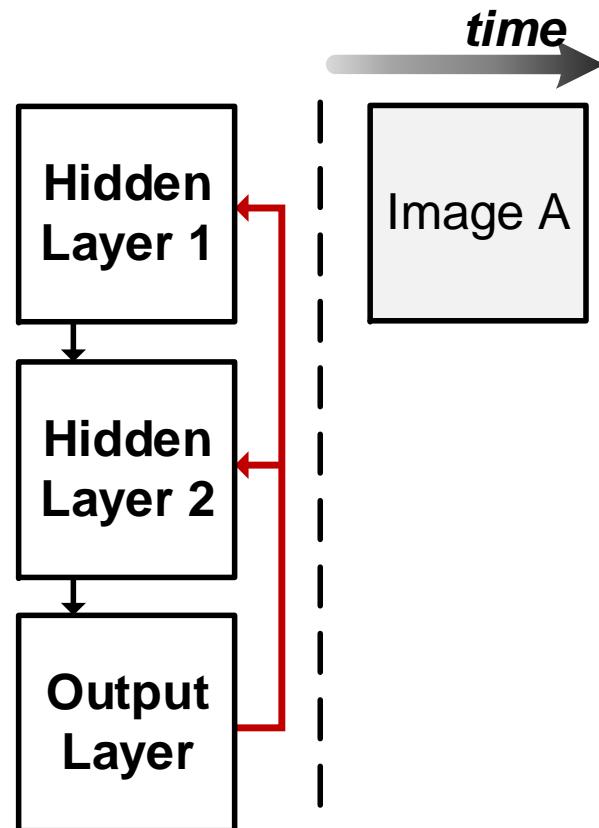
Conventional In-Order Updates

- ΔW values for an image updated **before** processing next image



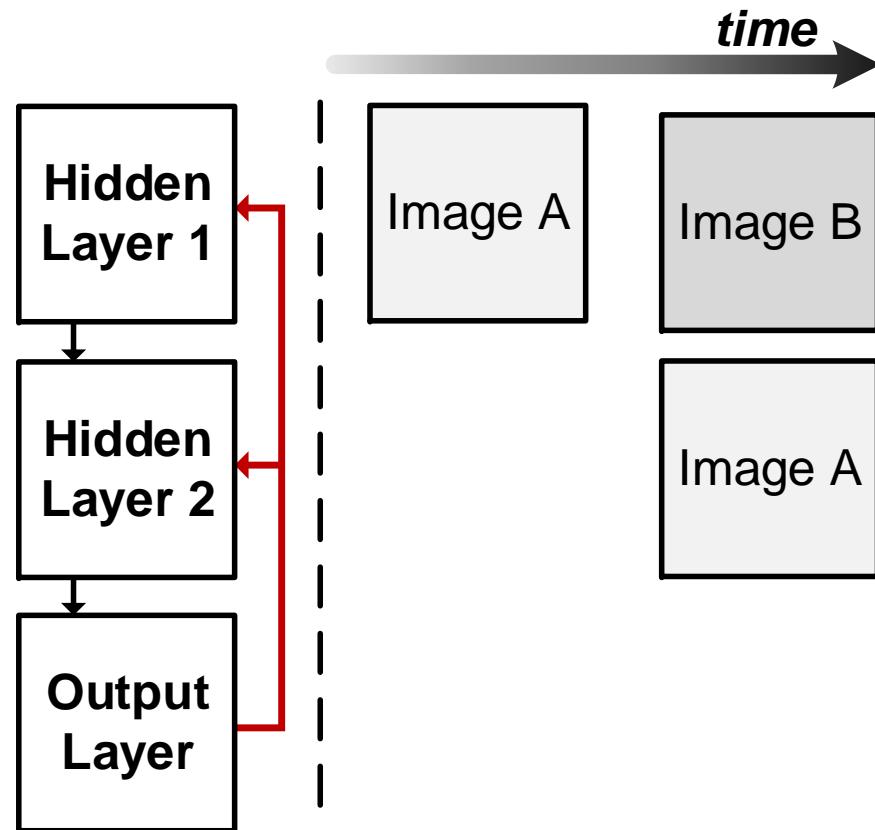
Implemented Out-of-Order Processing

- Process next image **before** updating ΔW for an image



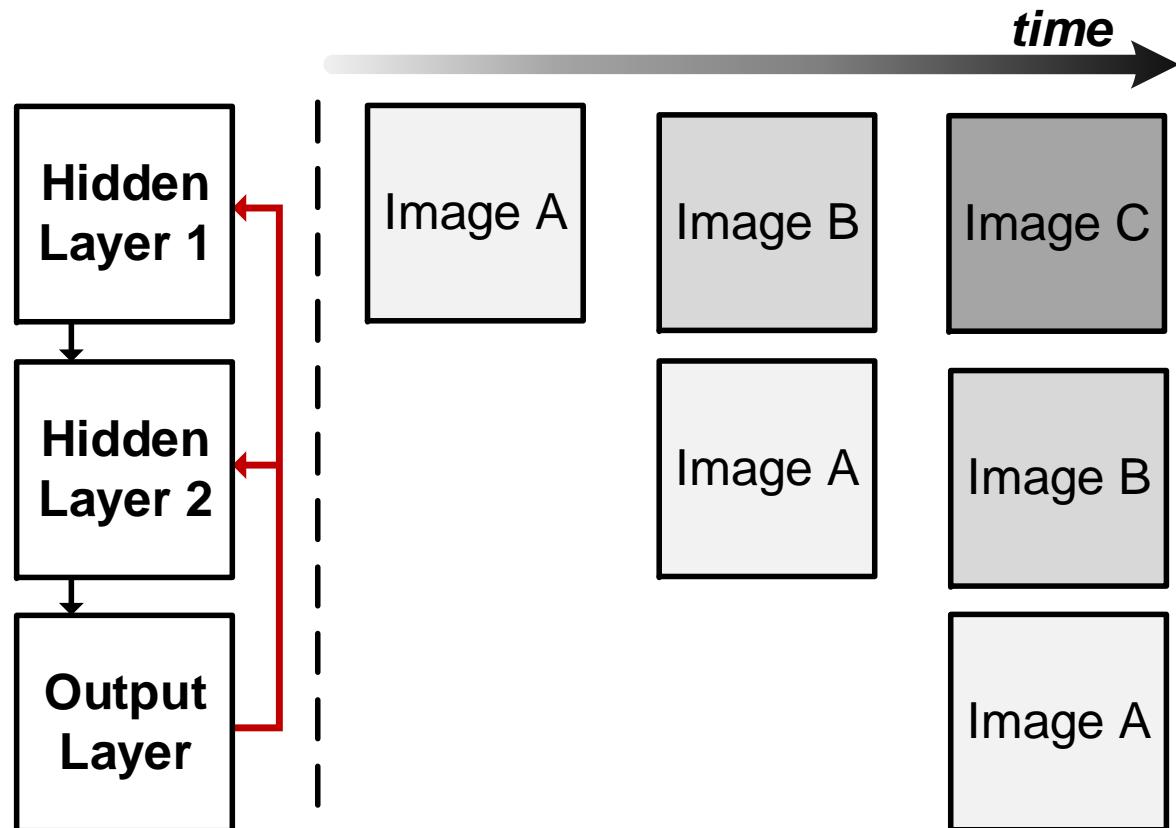
Implemented Out-of-Order Processing

- Process next image **before** updating ΔW for an image



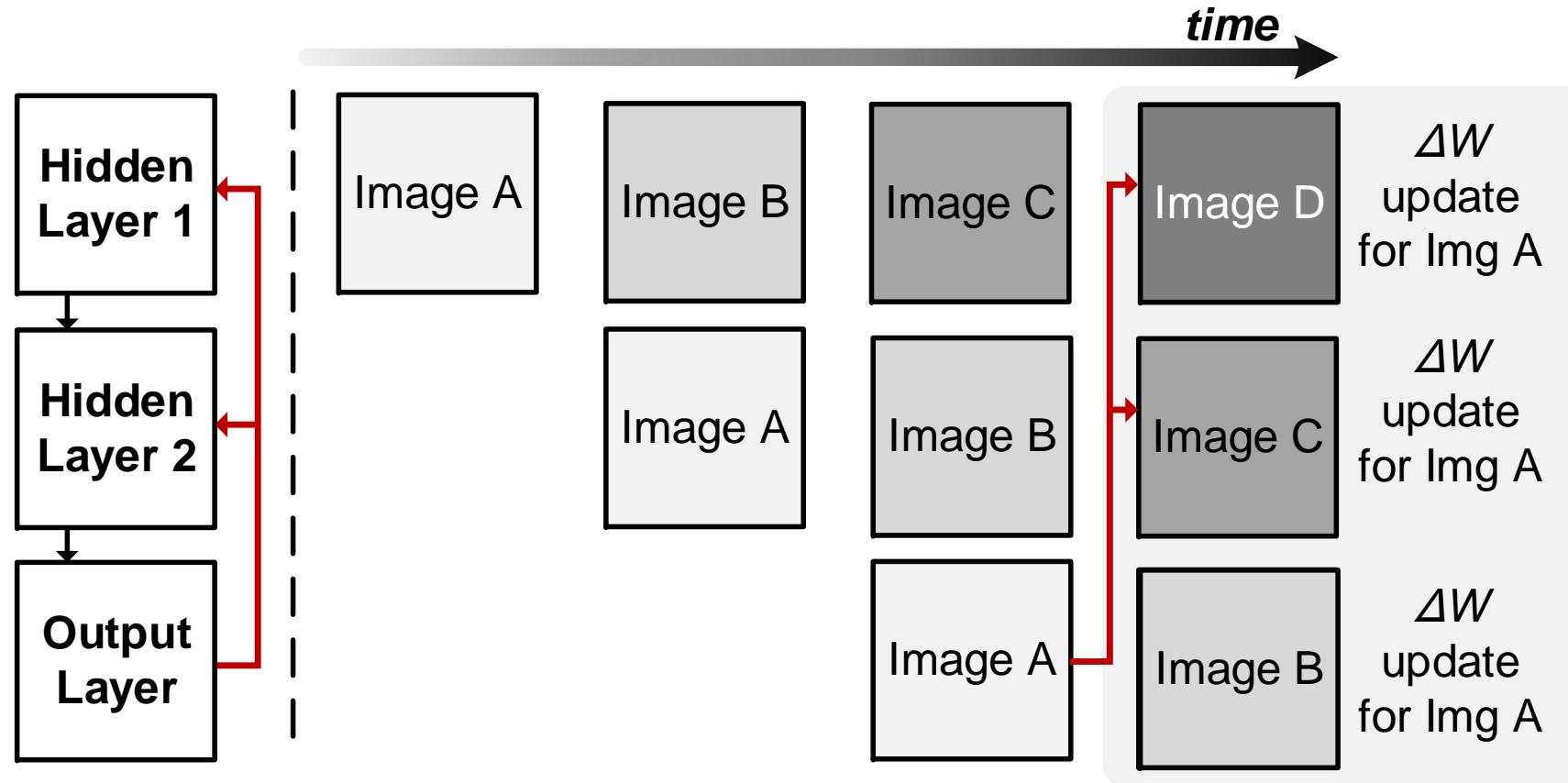
Implemented Out-of-Order Processing

- Process next image **before** updating ΔW for an image



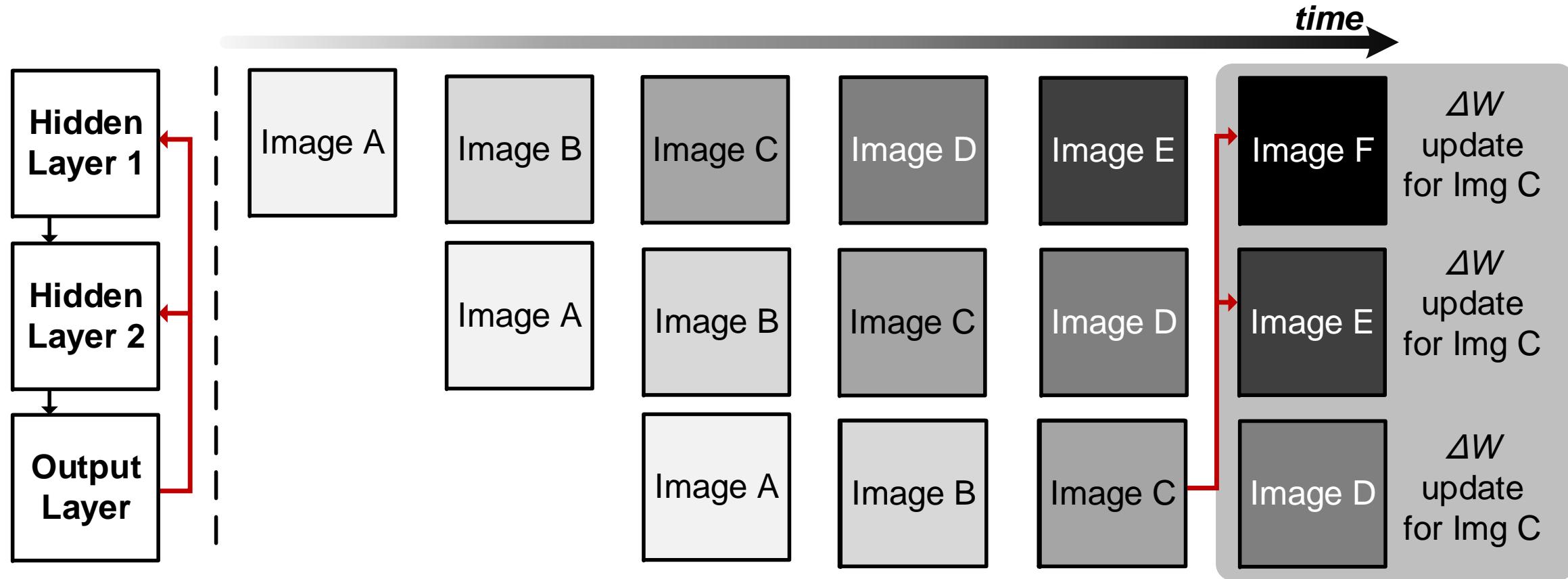
Implemented Out-of-Order Processing

- Process next image **before** updating ΔW for an image
- Simultaneously make updates for prior images



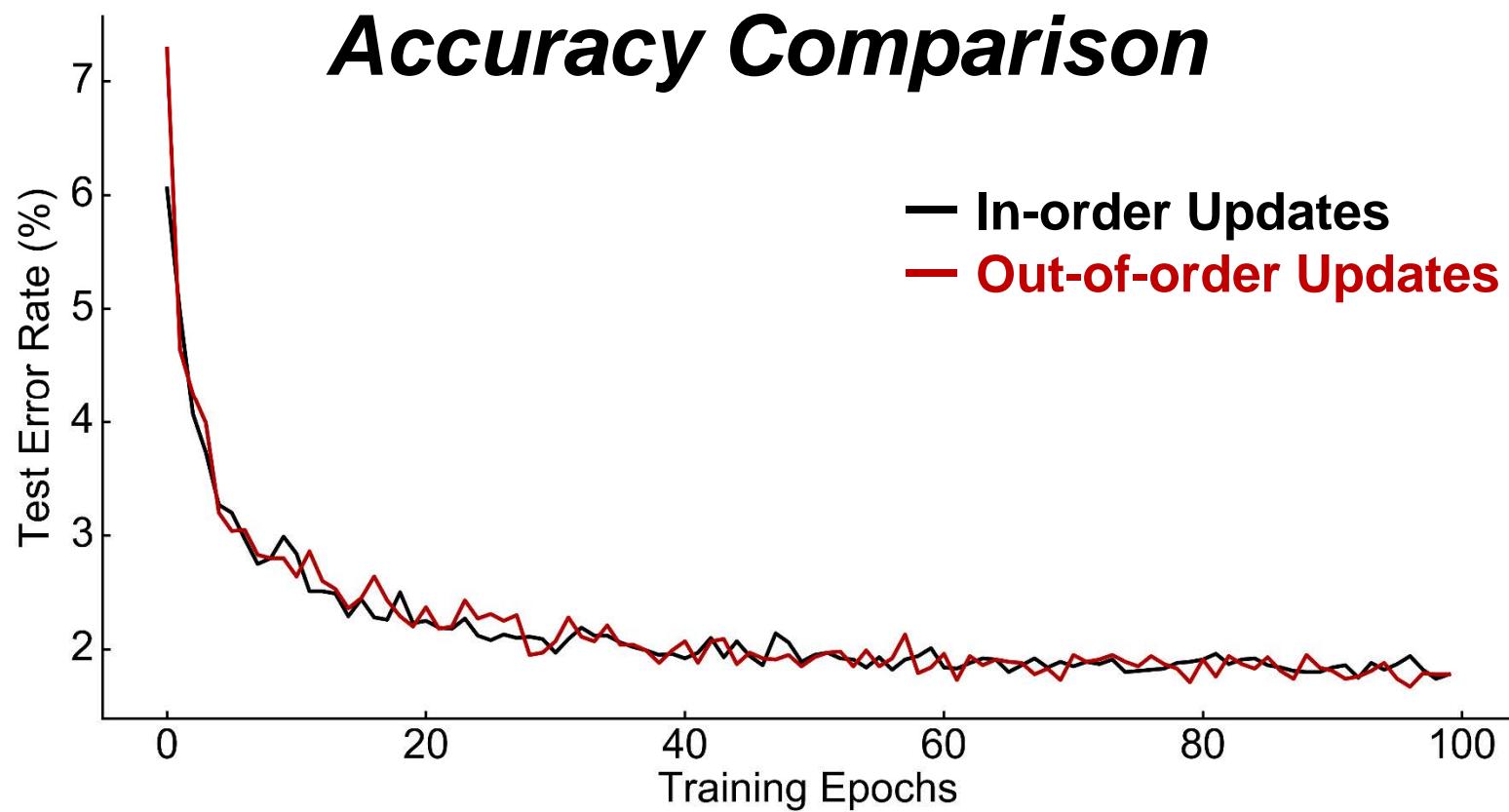
Implemented Out-of-Order Processing

- Increase throughput by $\times 4$, reduce number of memory reads
- Updates are gradual – before or after updates is not important



Statistical Performance

- Software simulations show no statistical degradation with out-of-order processing



Outline

I. Motivation

II. Neuromorphic Network

I. General Overview

II. Algorithm Modification

III. Hardware Implementation

I. Overall Architecture

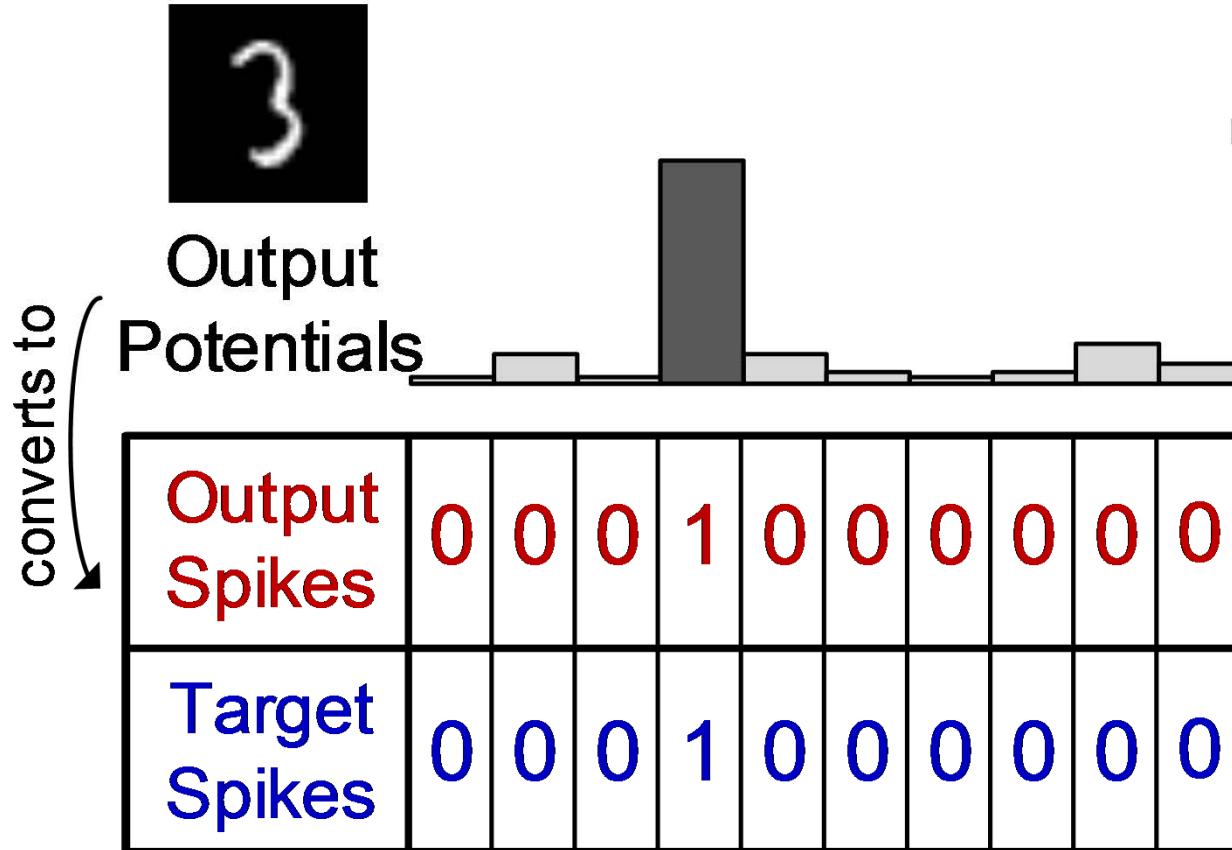
II. Out-of-order Weight Updates

III. Update Skipping Mechanism

IV. Measurements

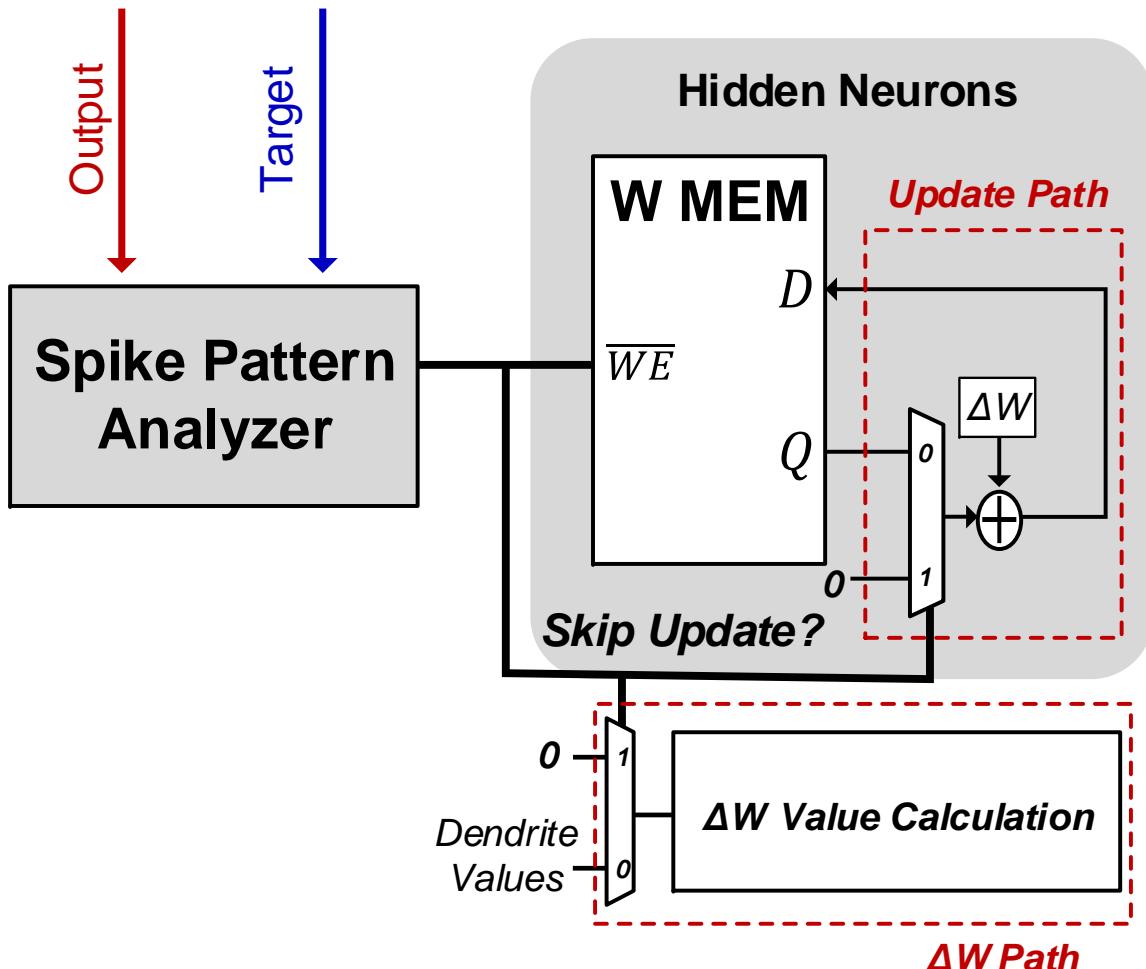
V. Conclusion

Observation for Update Rules



- Matching spike patterns
 - Target & forward spikes match
 - 0 difference in apical dendrites
 - $\Delta W = 0$ across all neurons
 - No need for updates!

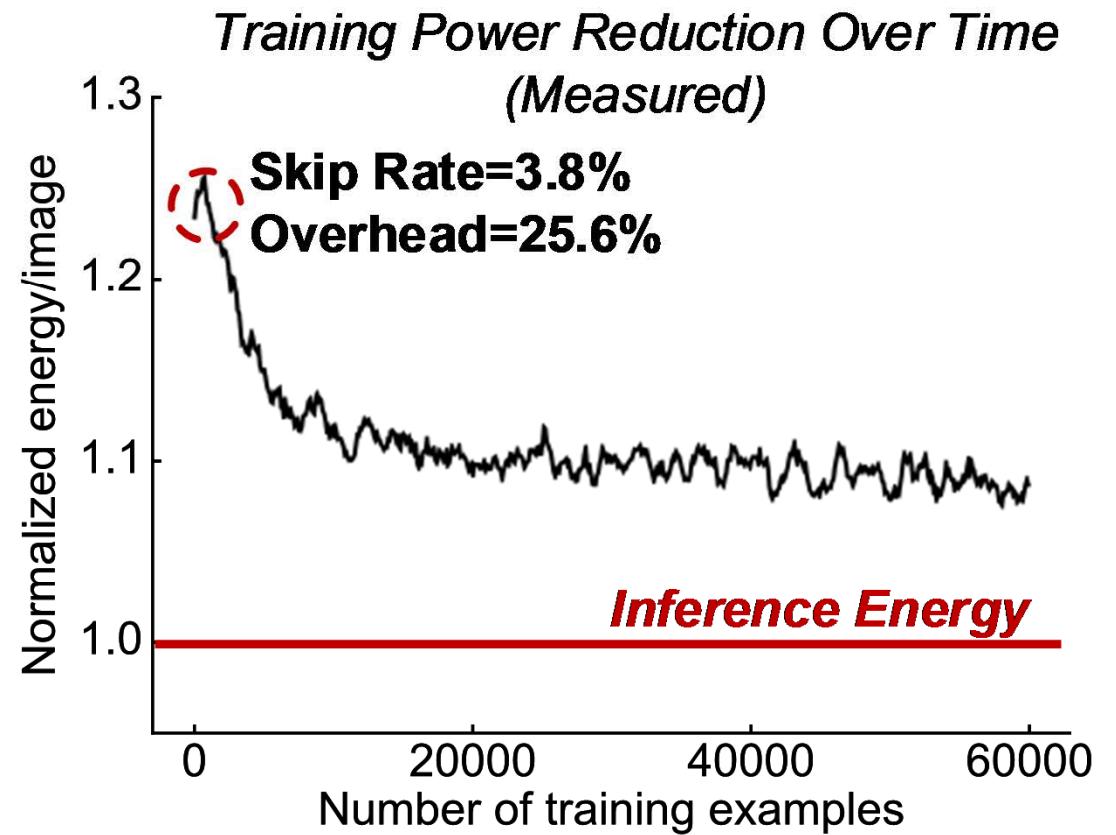
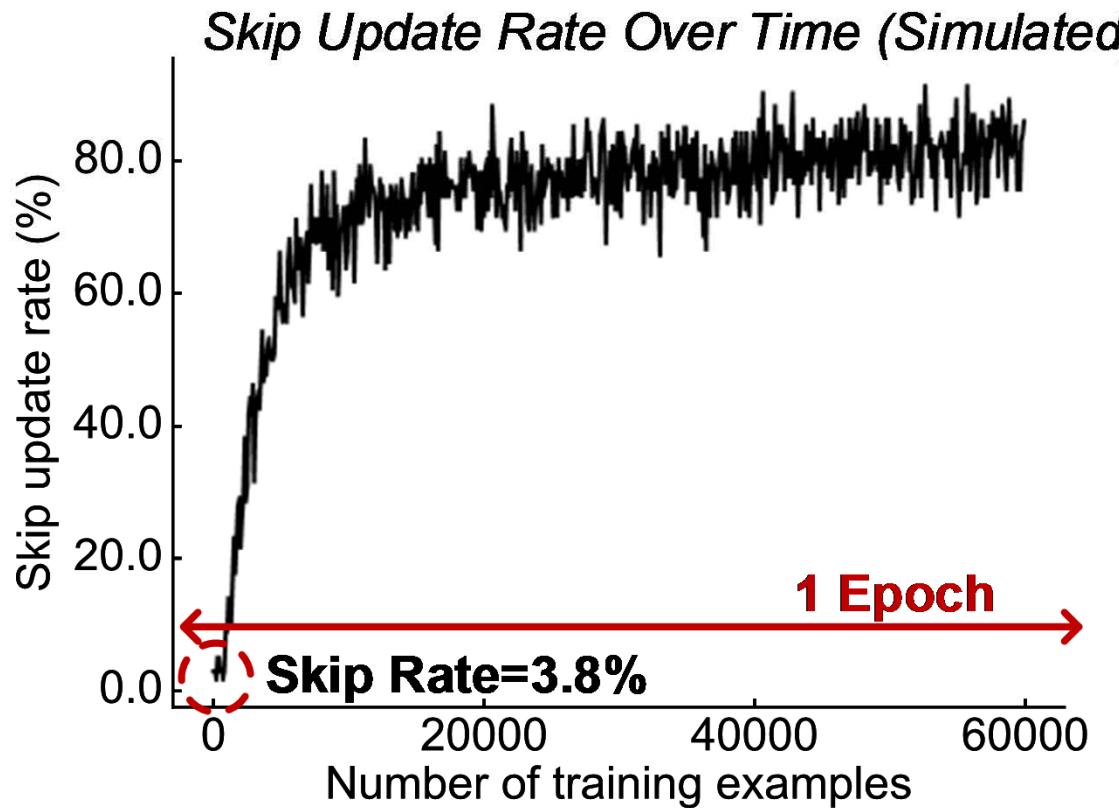
Skipping Updates for Matching Spikes



- Deserializes spike patterns and detect matching spikes
- With matching spikes:
 - Block inputs to ΔW calculation units
 - Set write enable low
- Saves switching & memory write energy

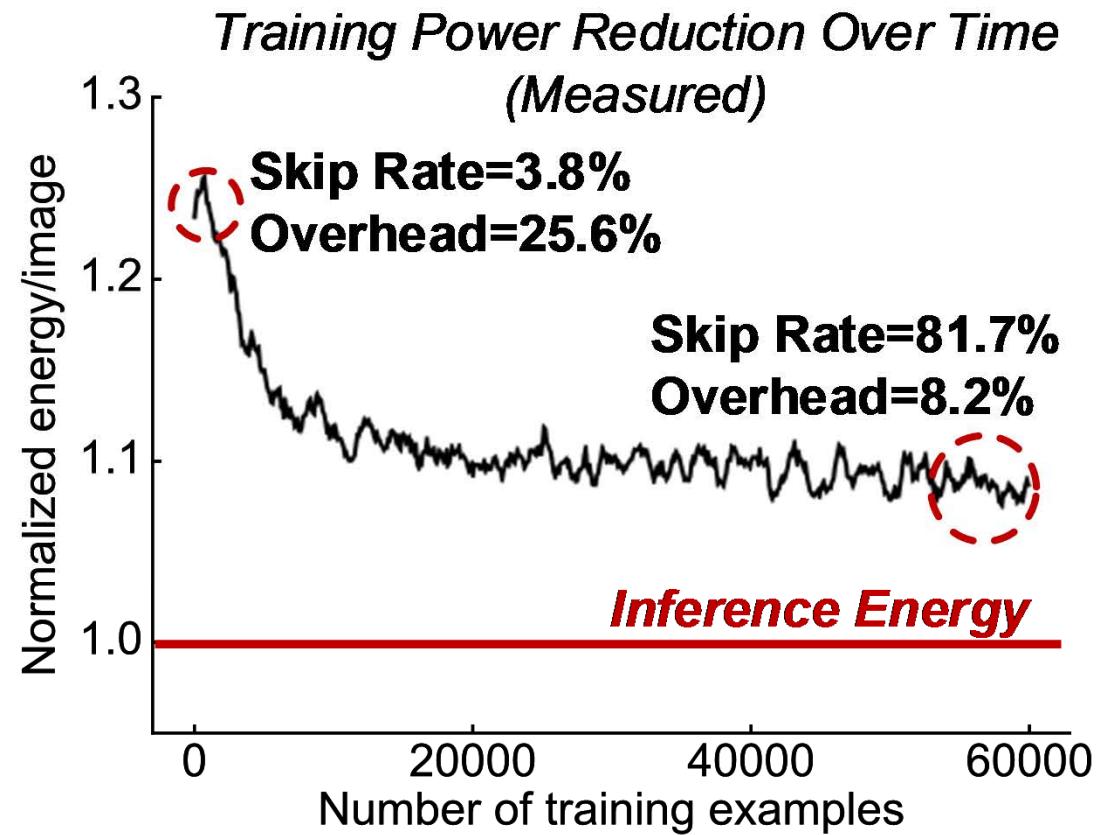
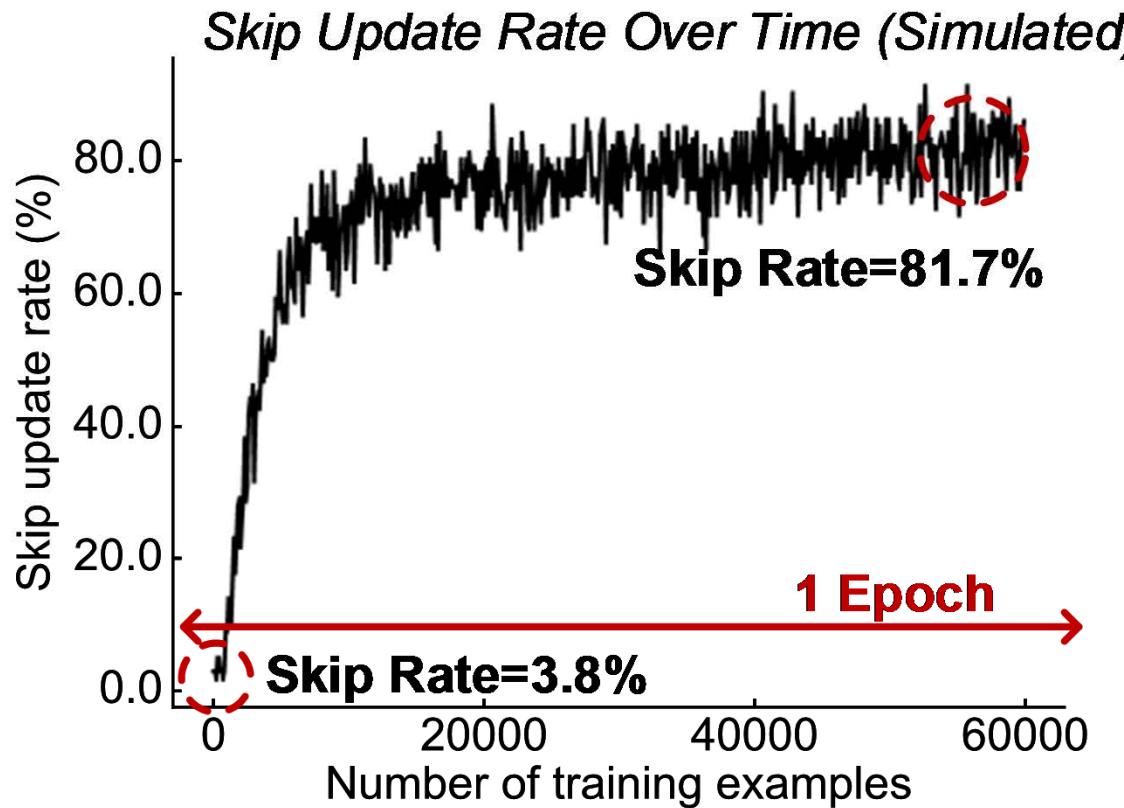
Measured Effect of Update Skipping

- Skip rate increases as training progresses



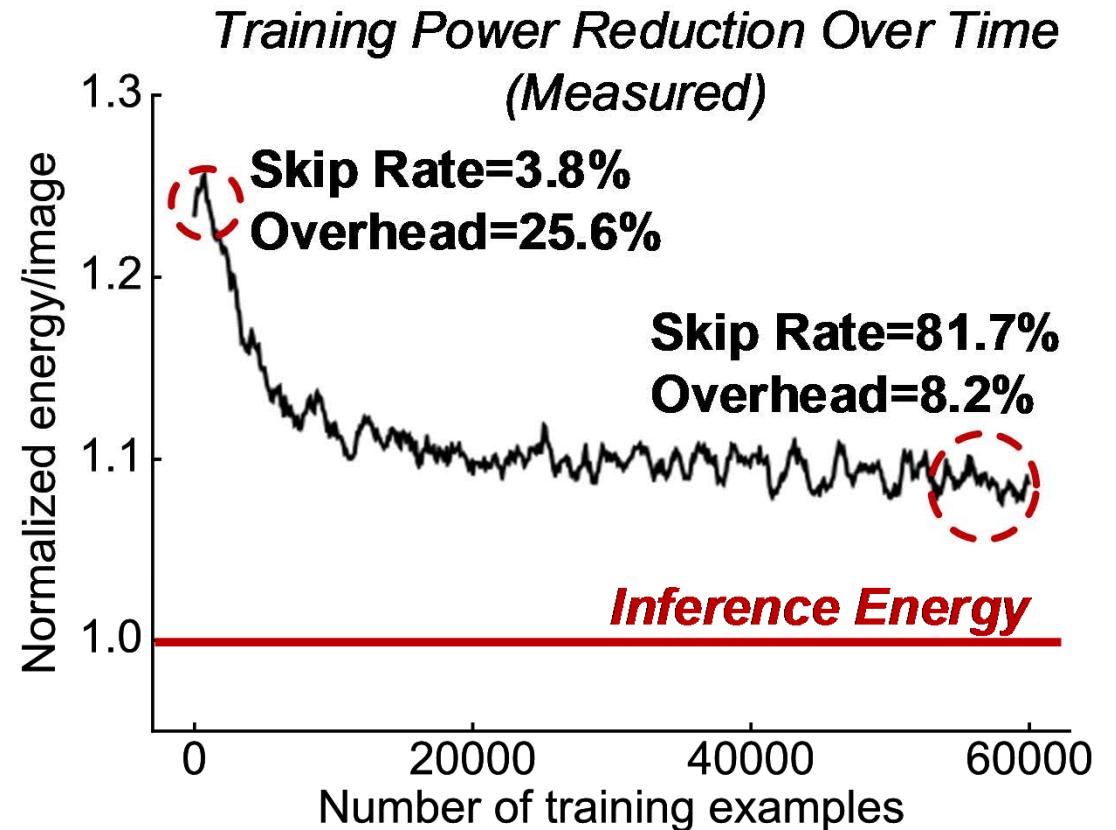
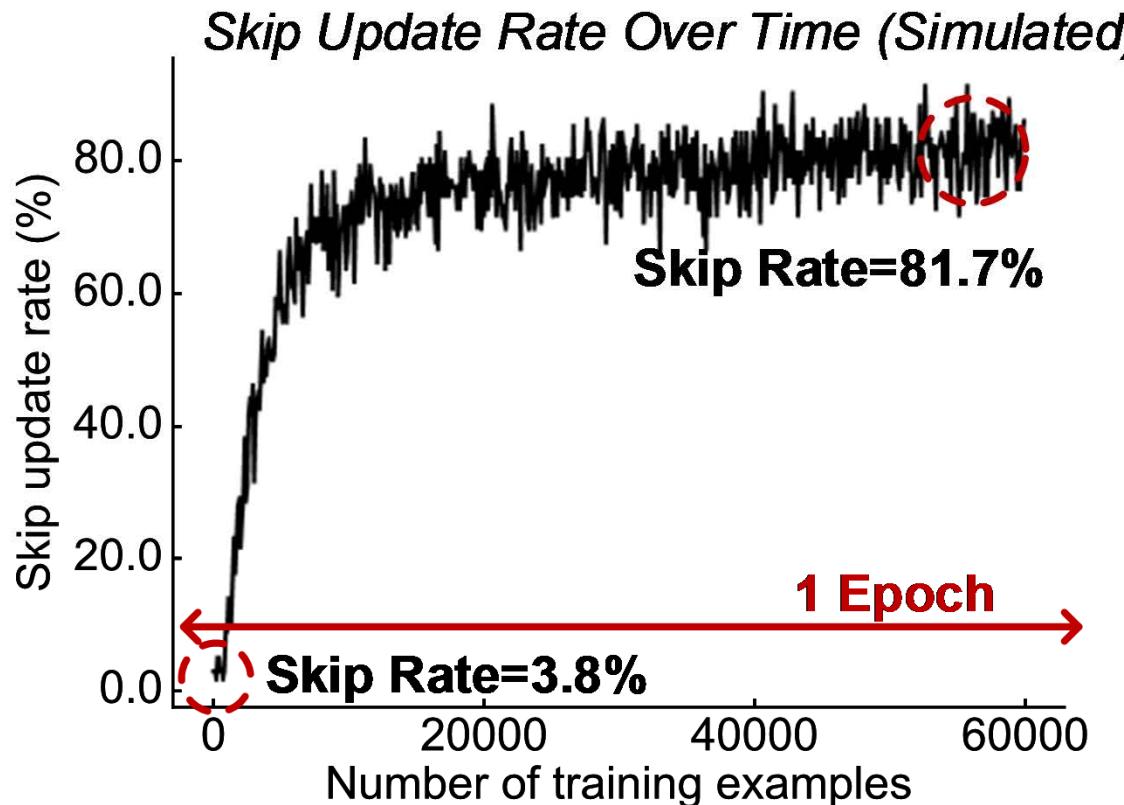
Measured Effect of Update Skipping

- Skip rate increases as training progresses



Measured Effect of Update Skipping

- Skip rate increases as training progresses



- After 100 epochs: **Skip Rate=94.5%, Energy Overhead=7.5%**

Outline

I. Motivation

II. Neuromorphic Network

I. General Overview

II. Algorithm Modification

III. Hardware Implementation

I. Overall Architecture

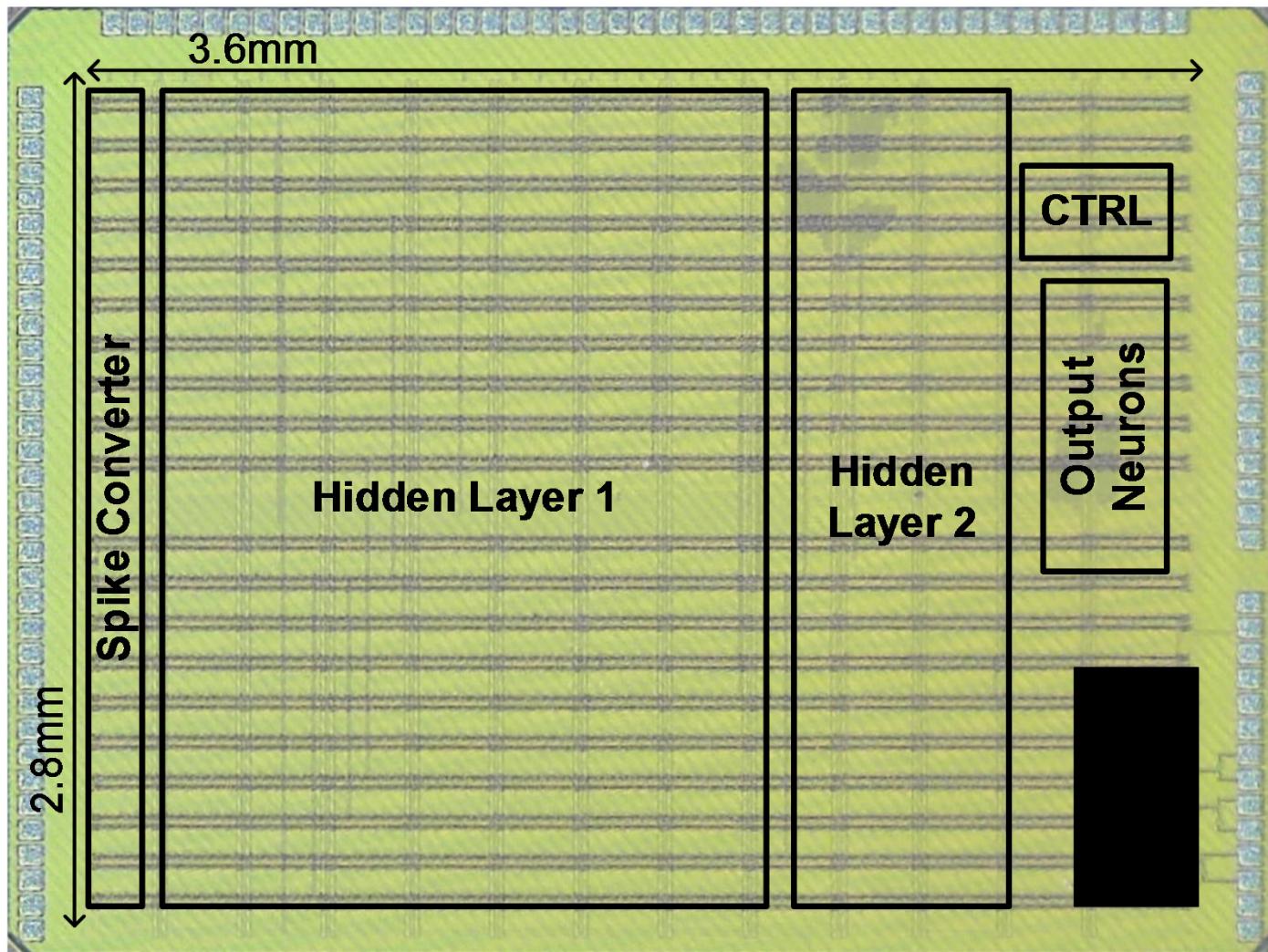
II. Out-of-order Weight Updates

III. Update Skipping Mechanism

IV. Measurements

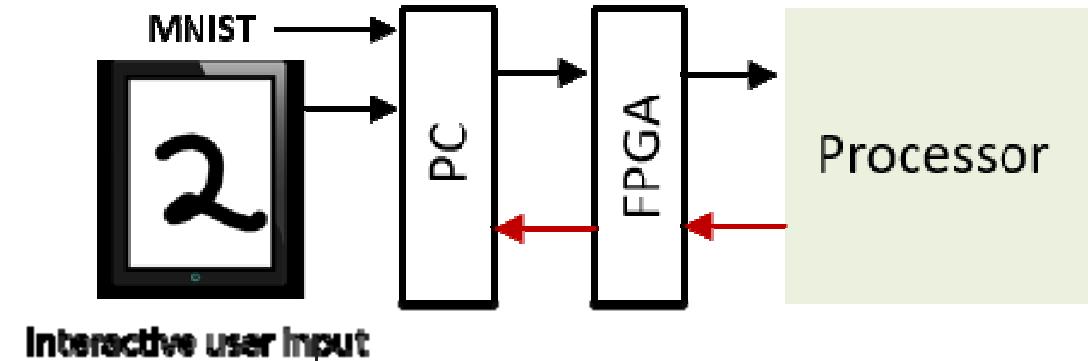
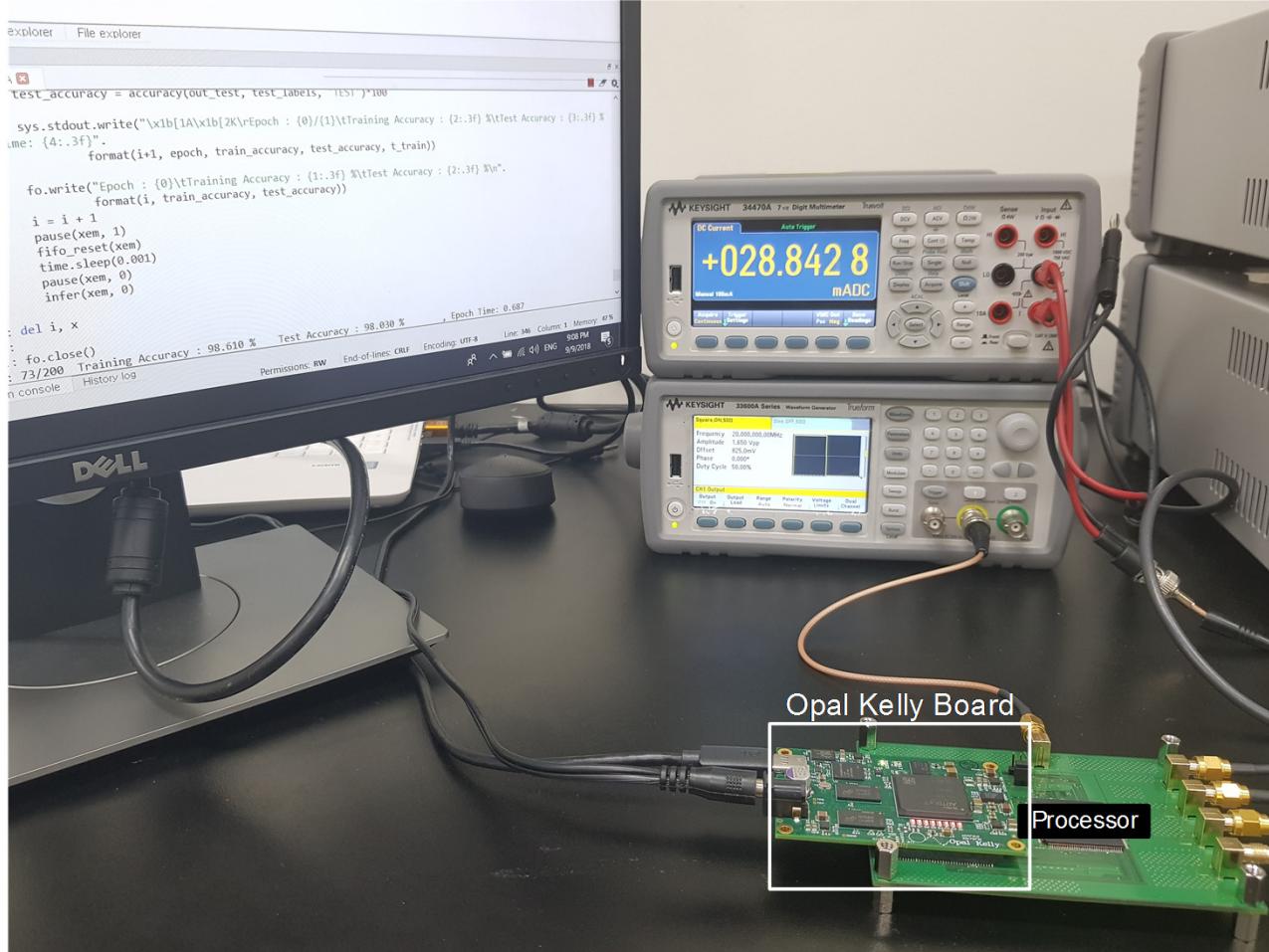
V. Conclusion

Die Photograph & Summary Table



	Specification
Technology	65nm LP
V_{dd}	0.8V
Power	23.1mW(Train) 23.6mW(Infer)
Frequency	20MHz
Efficiency	3.42TOPS/W
Core Area	$2.8 \times 3.6\text{mm}^2$
Throughput	94K img/s(Train) 100K img/s(Infer)

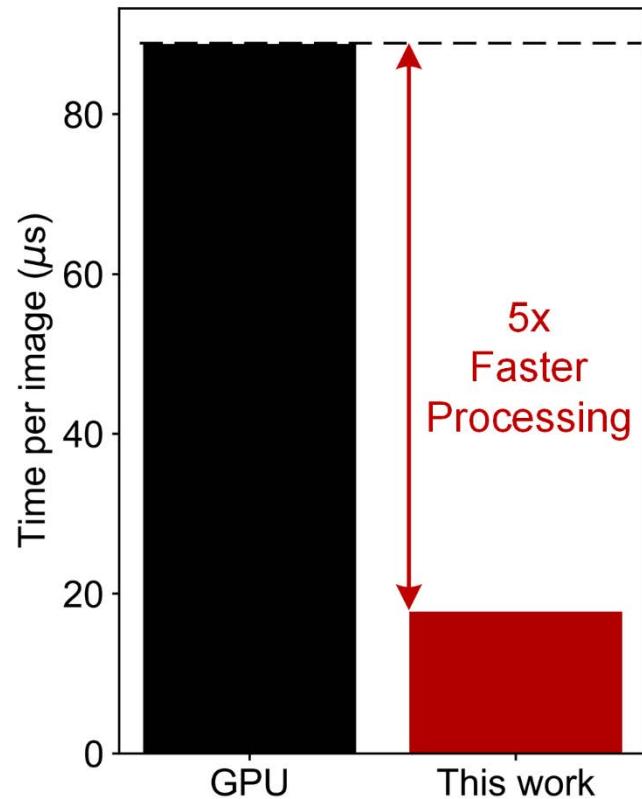
Measurement Setup



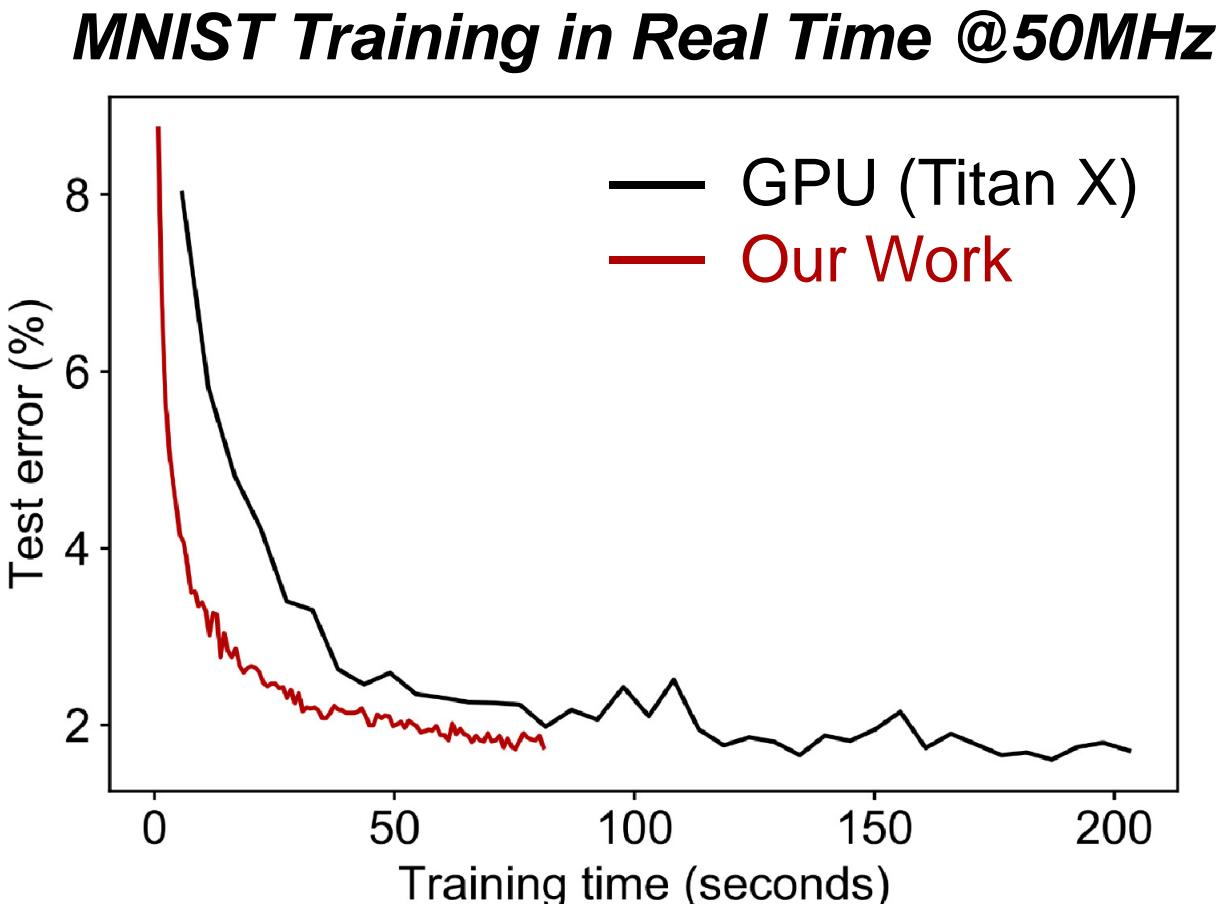
- Achieves 97.8% accuracy after training

High Performance Training

- Training shows high performance compared to GPU

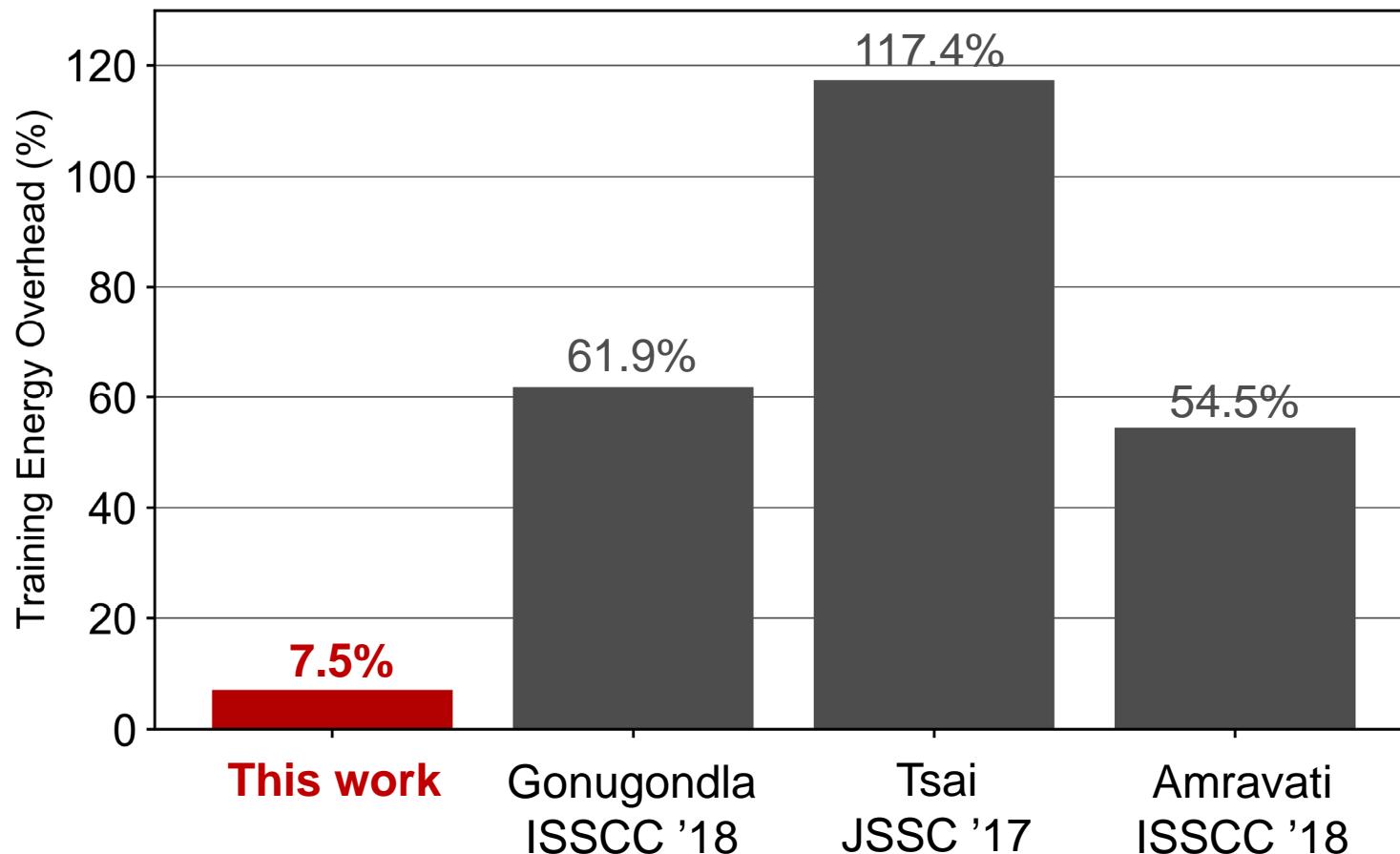


Baseline: 784-200-200-10 MLP
SGD with batch size = 64
Nvidia Titan X



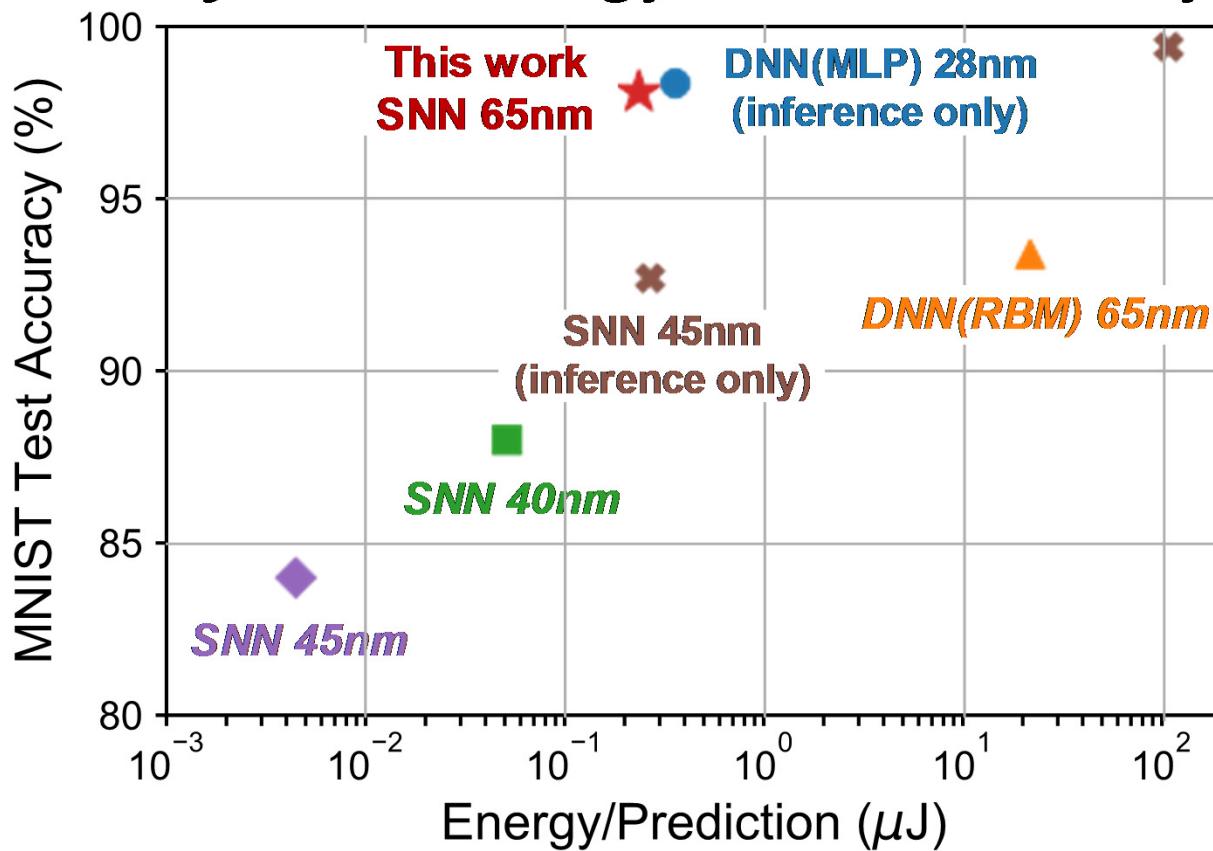
Efficient Training

- Exploit properties of **spike-based training** for low training overhead



Efficient Inference

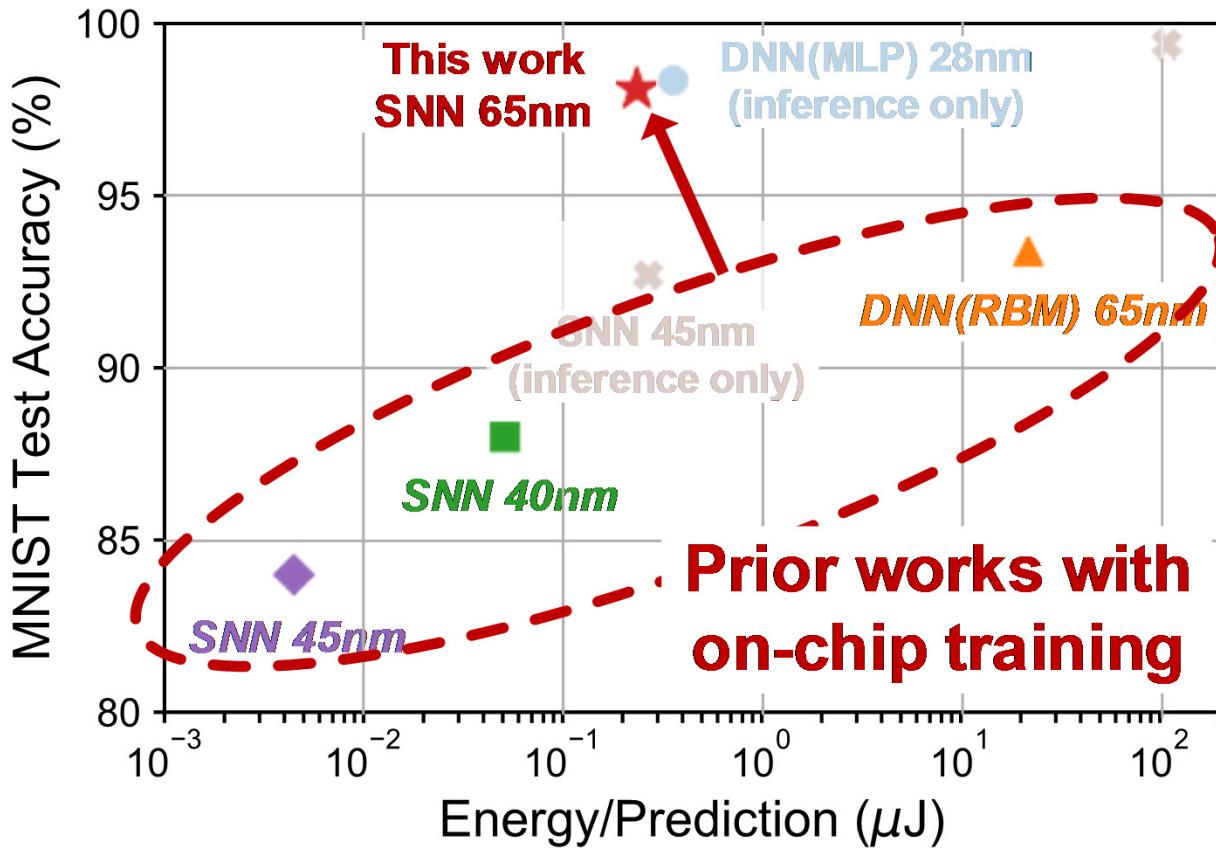
Inference Energy vs. Test accuracy



- 236.5nJ/image for inference, 254.3nJ/image for training
- 97.83% accuracy on MNIST test dataset

Efficient Inference

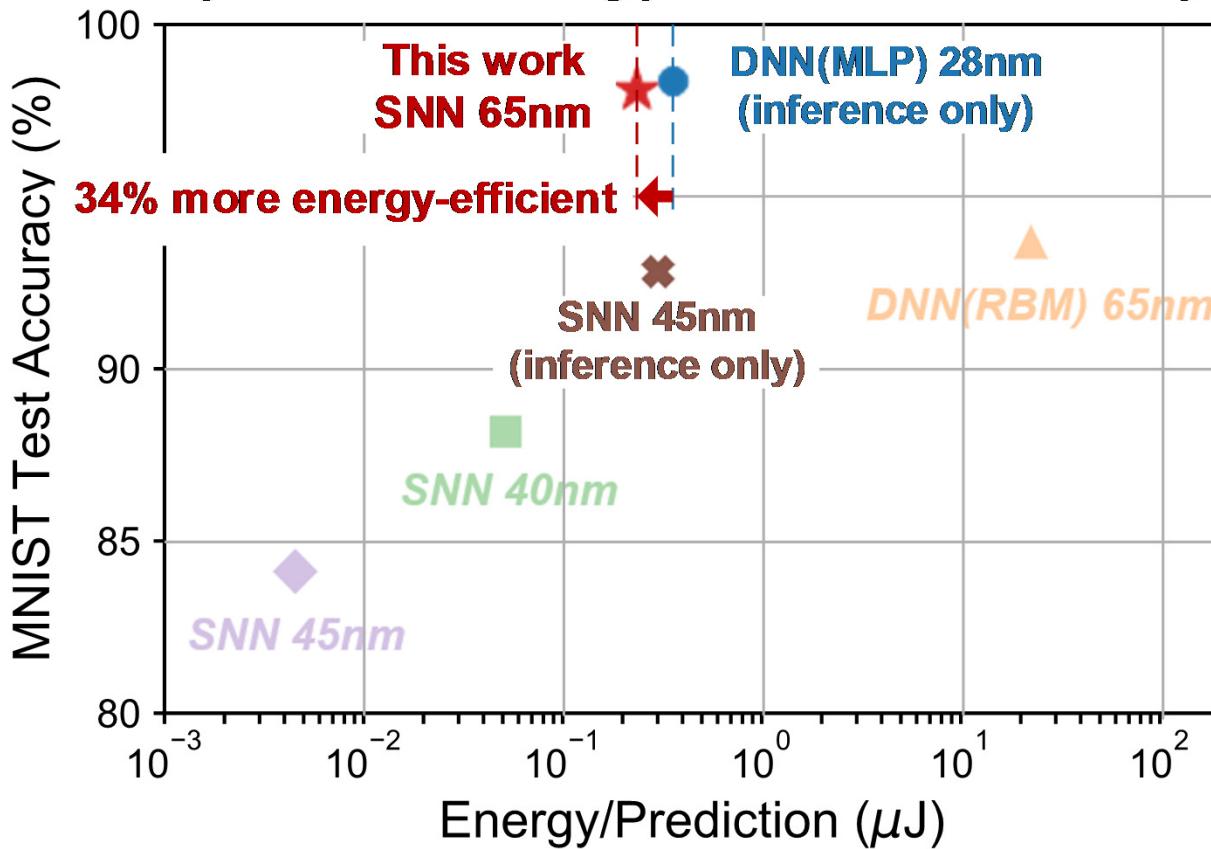
Inference Energy vs. Test accuracy



- 236.5nJ/image for inference, 254.3nJ/image for training
- 98.73% accuracy on MNIST test dataset
- Breaks energy-accuracy tradeoff in previous on-chip training systems

Efficient Inference

Inference Energy vs. Test accuracy



- 236.5nJ/image for inference, 254.3nJ/image for training
- 98.73% accuracy on MNIST test dataset
- Breaks energy-accuracy tradeoff in previous on-chip training systems
- **34% less energy than DNN inference machine!**

Outline

I. Motivation

II. Neuromorphic Network

I. General Overview

II. Algorithm Modification

III. Hardware Implementation

I. Overall Architecture

II. Out-of-order Weight Updates

III. Update Skipping Mechanism

IV. Measurements

V. Conclusion

Conclusion

- Proposed neuromorphic network provides energy-efficient training
 - Hardware friendly neuromorphic algorithm
 - Out-of-order weight updates
 - Update skipping mechanism
- Achieved design goals
 - Good energy efficiency compared to inference-only DNN machines (34% lower)
 - Low training energy overhead (7.5% over inference)
 - Classification performance near back-propagation (97.83%)

LNPU: A 25.3 TFLOPS/W Sparse Deep-Neural-Network Learning Processor with Fine-Grained Mixed Precision of FP8-FP16

**Jinsu Lee, Juhyoung Lee, Donghyeon Han,
Jinmook Lee, Gwangtae Park, and Hoi-Jun Yoo**

**Semiconductor System Lab.
School of EE, KAIST**

Outline

I. Introduction

II. Overall Architecture

III. Key Features

- 1) **Fully Reconfigurable Sparse DL Core for High Throughput**
- 2) **Input Load Balancer for High PE Utilization**
- 3) **FP8/FP16 Configurable PE for High Energy-Efficiency**

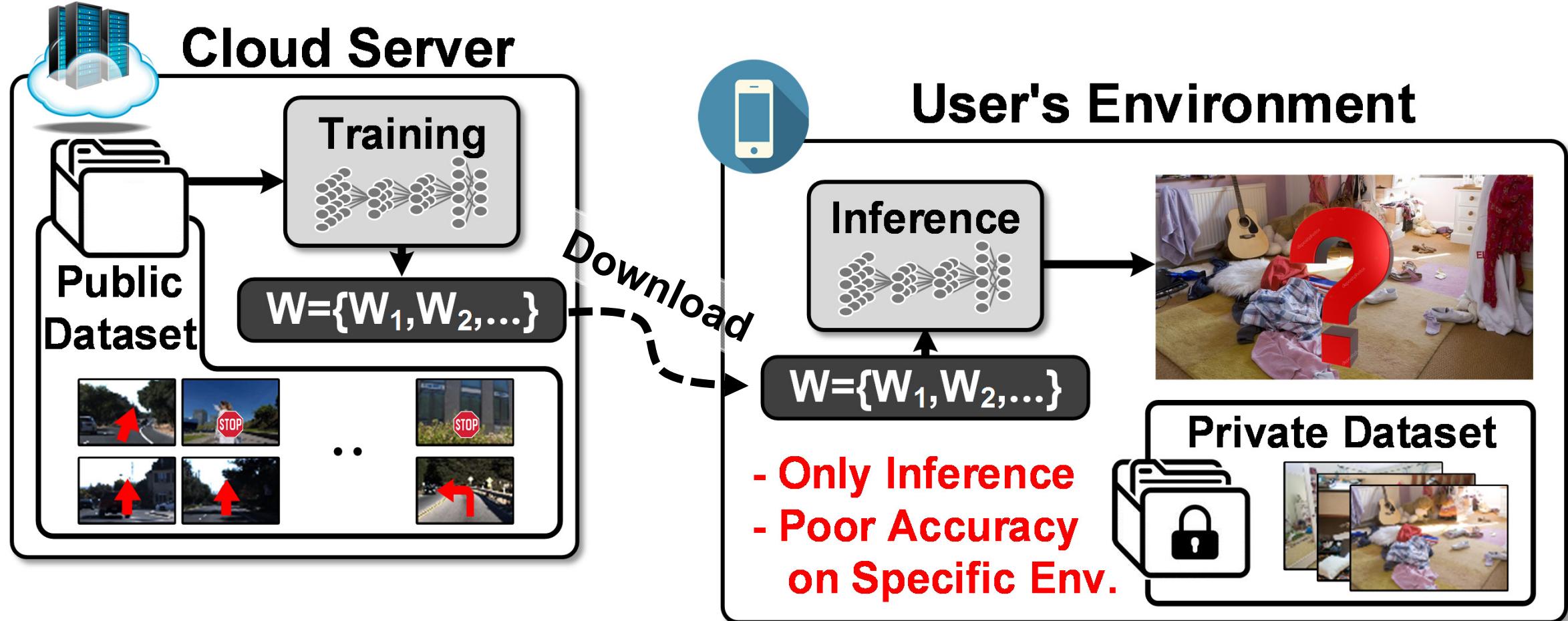
IV. Implementation Results

V. Verification System

VI. Conclusion

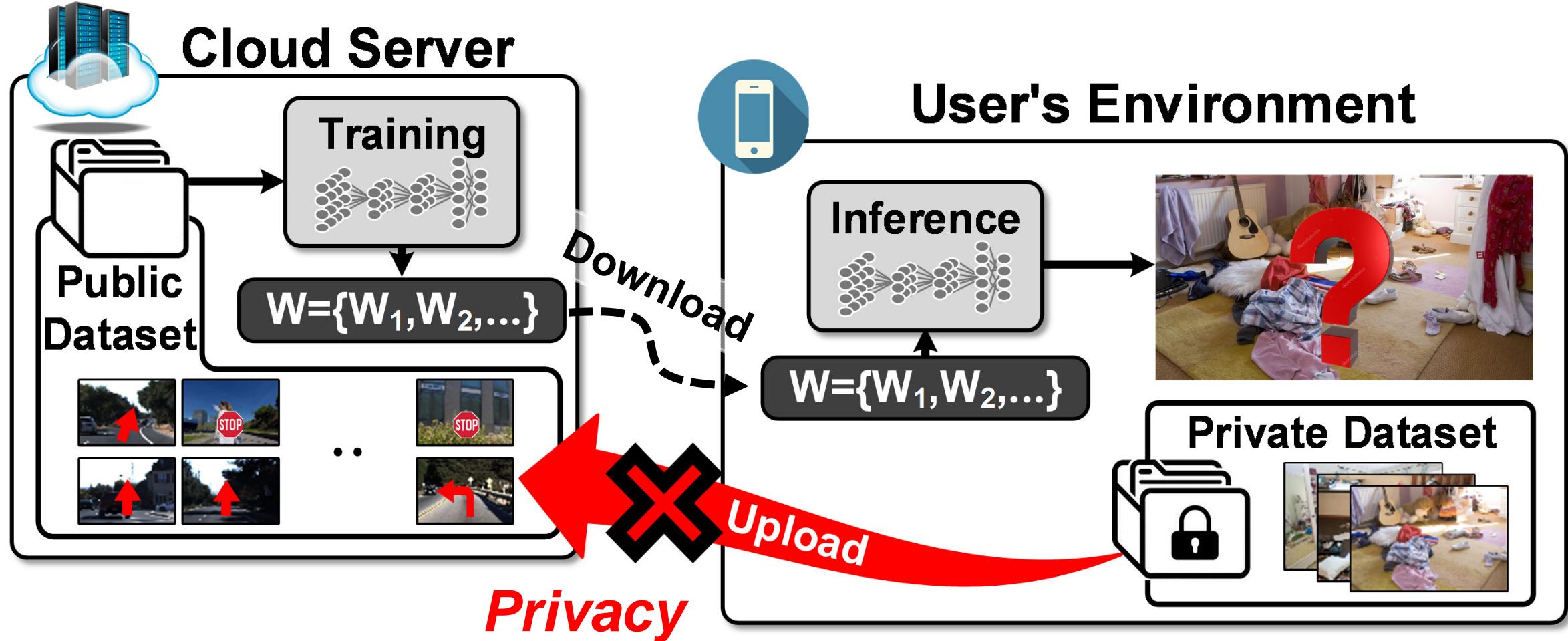
Training on Edge Devices

- DNN Inference is being Moved to Edge/Mobile Devices



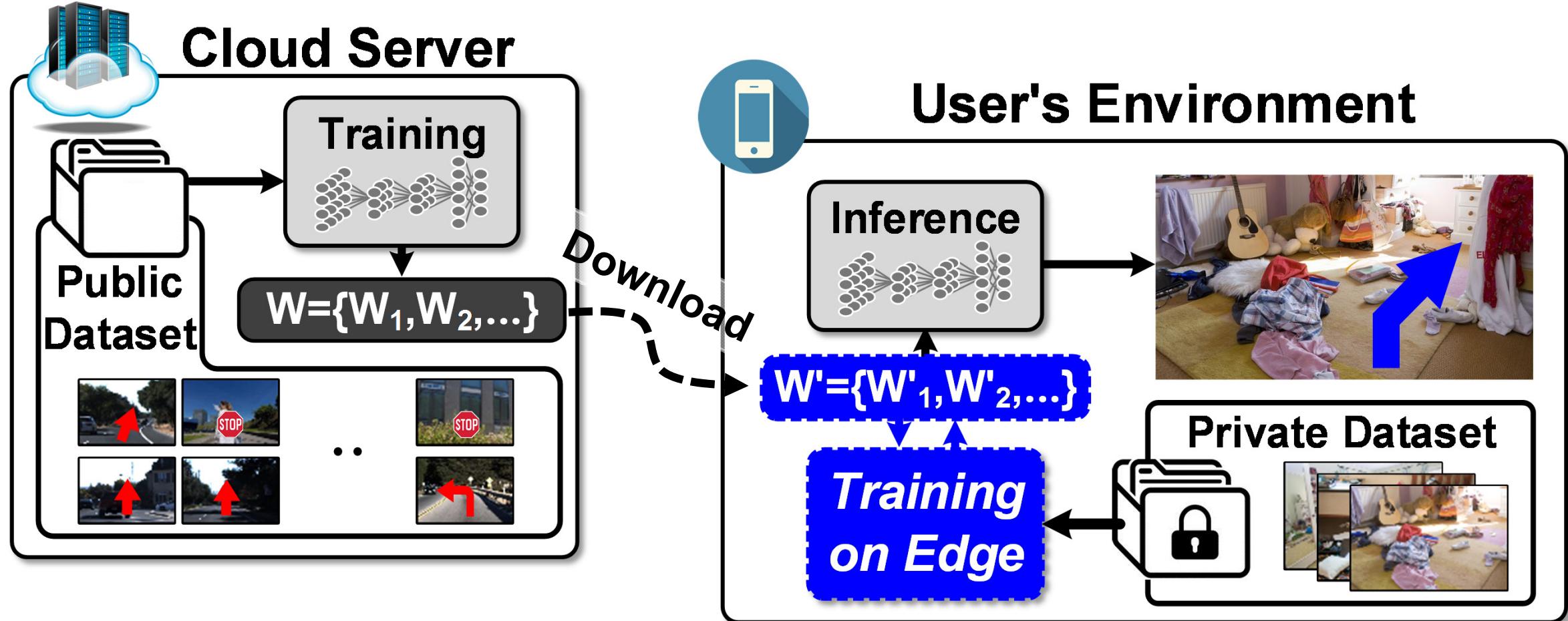
Training on Edge Devices

- User's Private Dataset Can't be Uploaded to Cloud Server



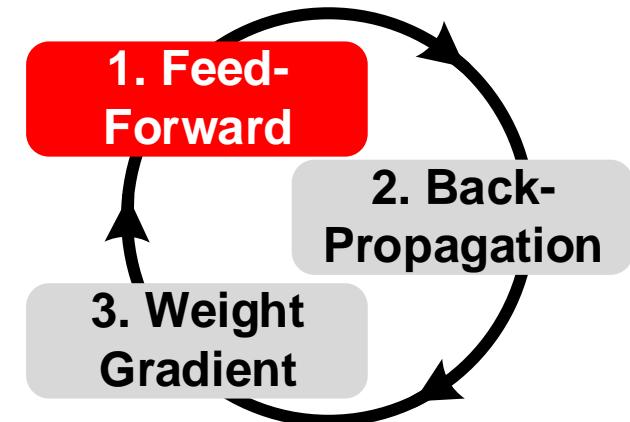
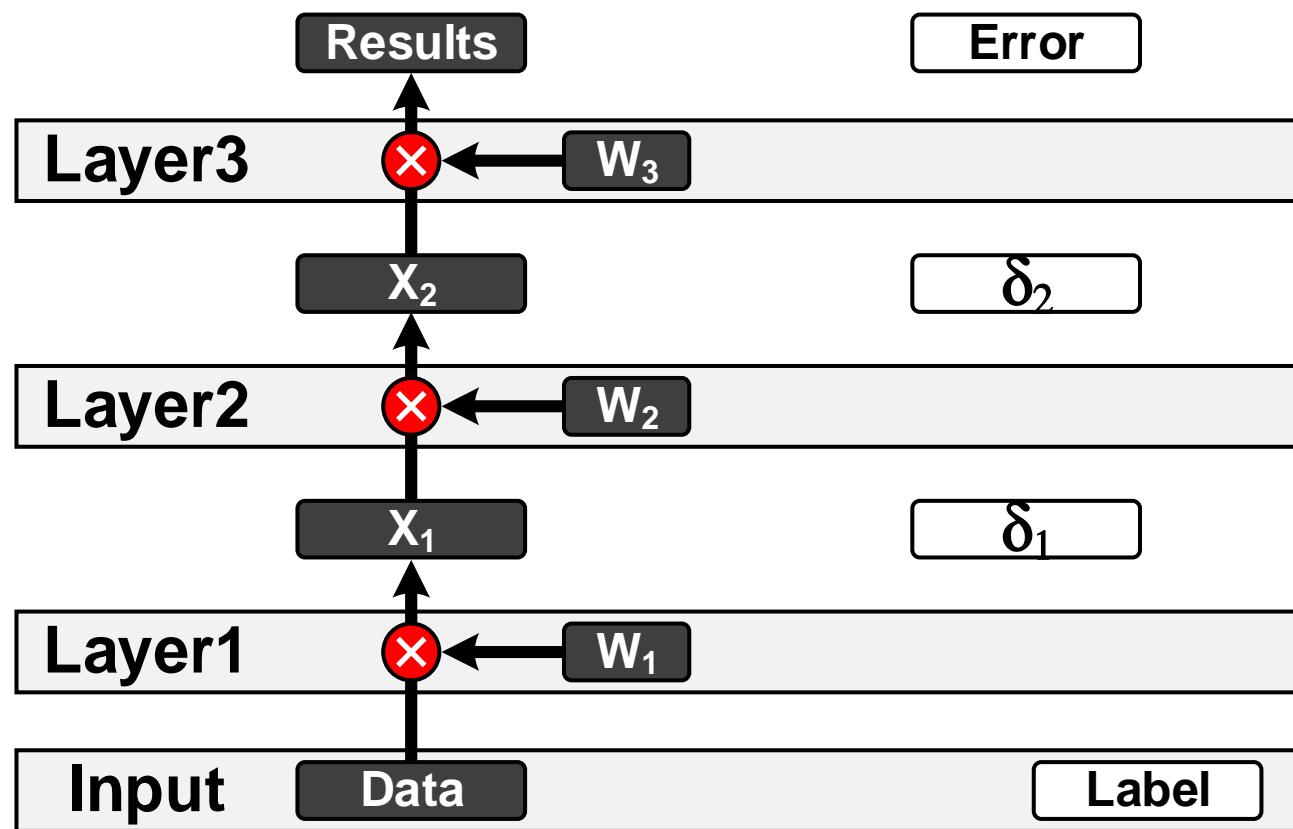
Training on Edge Devices

- On-Device Training for High Accuracy and Personalization

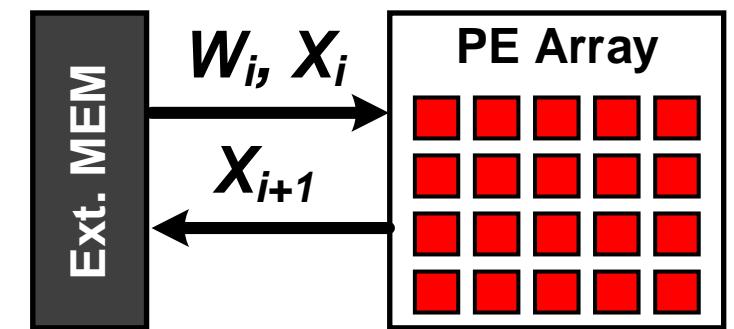


DNN Training Basics

1. Feed-Forward (FF) / Inference

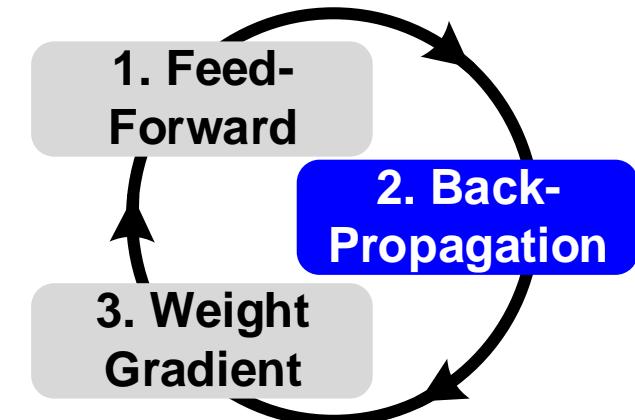
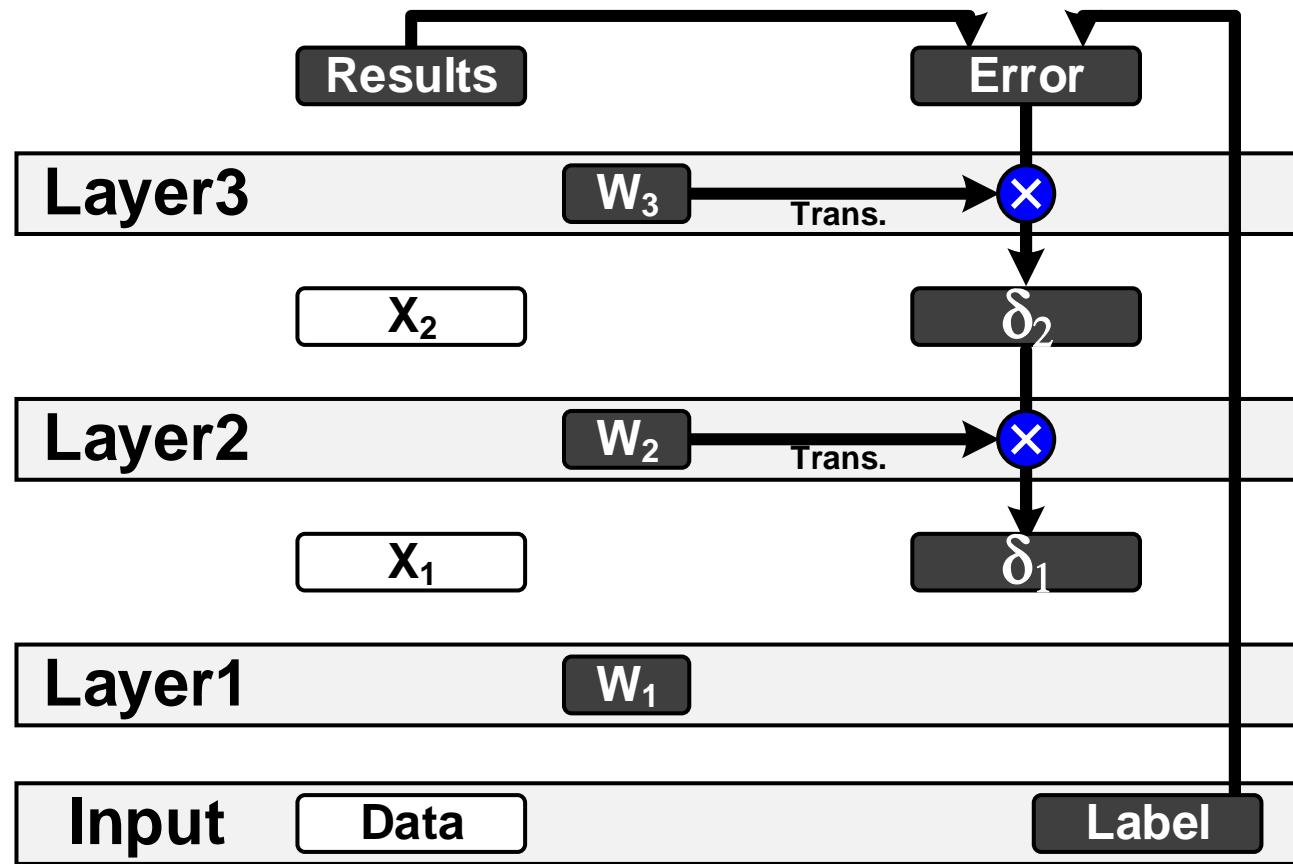


$$X_{i+1} = \sum_{ch_In} X_i * W_i + b$$

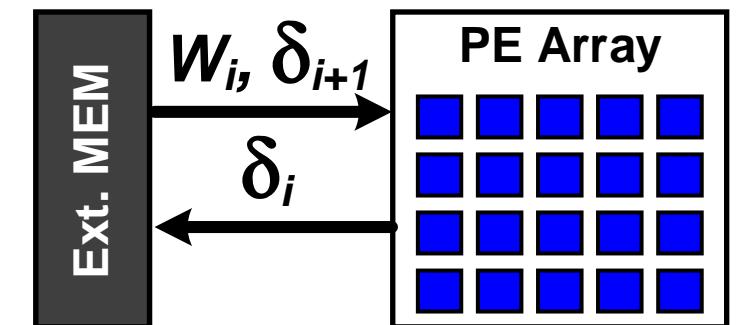


DNN Training Basics

2. Back-Propagation (BP)

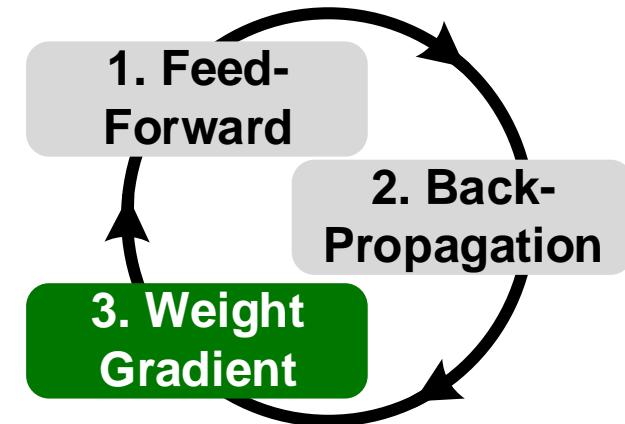
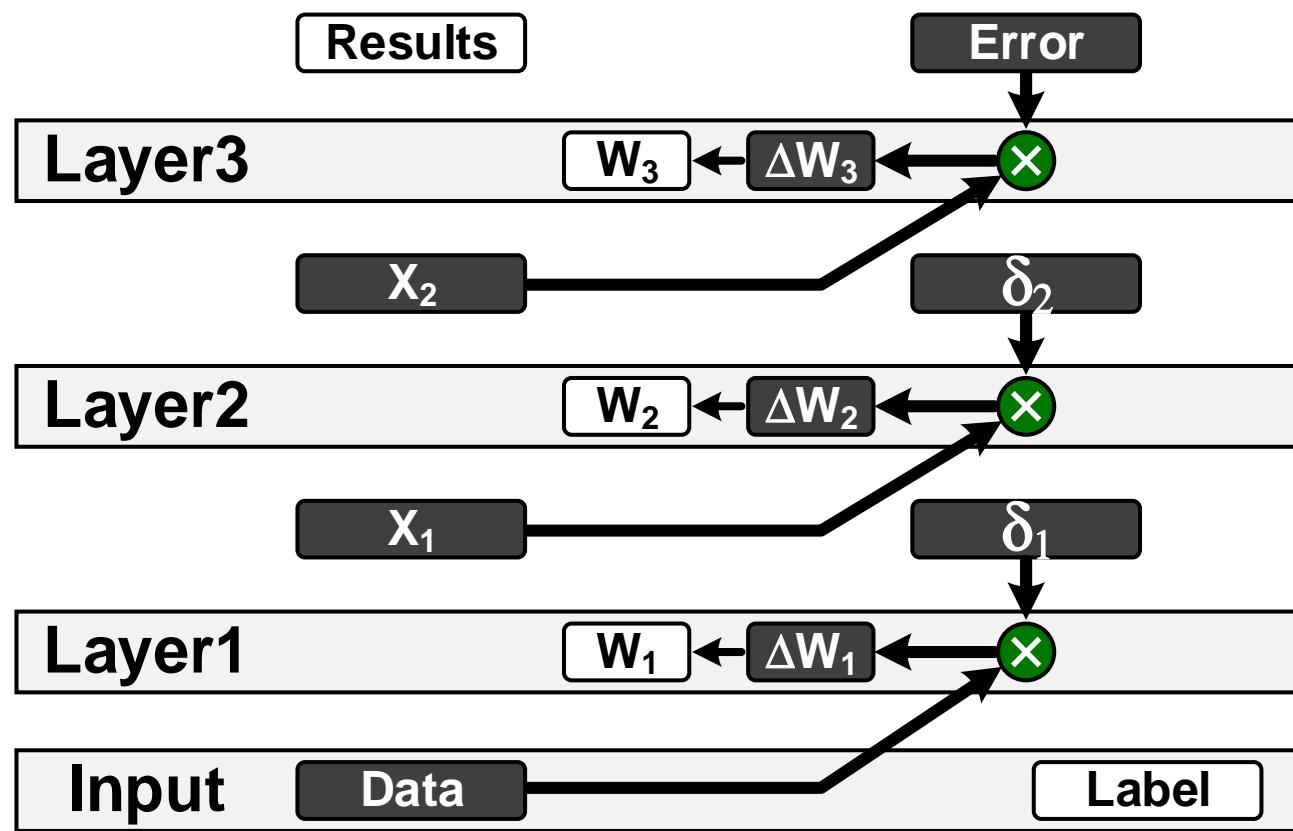


$$\delta_i = \sum_{Ch_Out} \delta_{i+1} * W_i^T$$

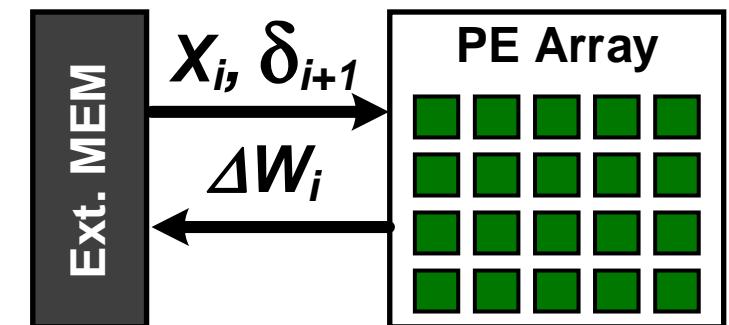


DNN Training Basics

3. Weight Gradient Update (WG)



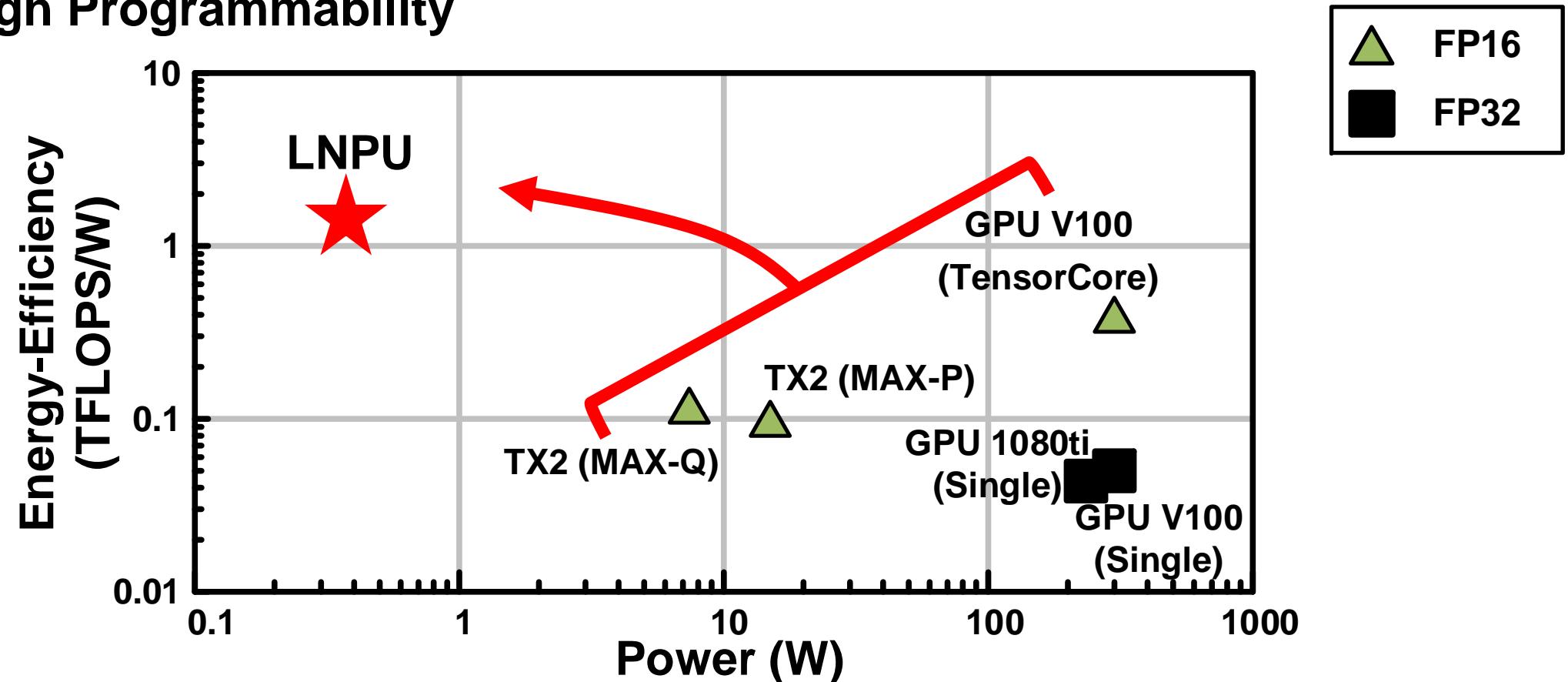
$$\Delta W_i = \sum_{\text{Image or Batch}} X_i * \delta_{i+1}$$



DNN Training HW

GPU: High Power Consumption & Low Energy-Efficiency

- Floating Point ALU
- High Programmability

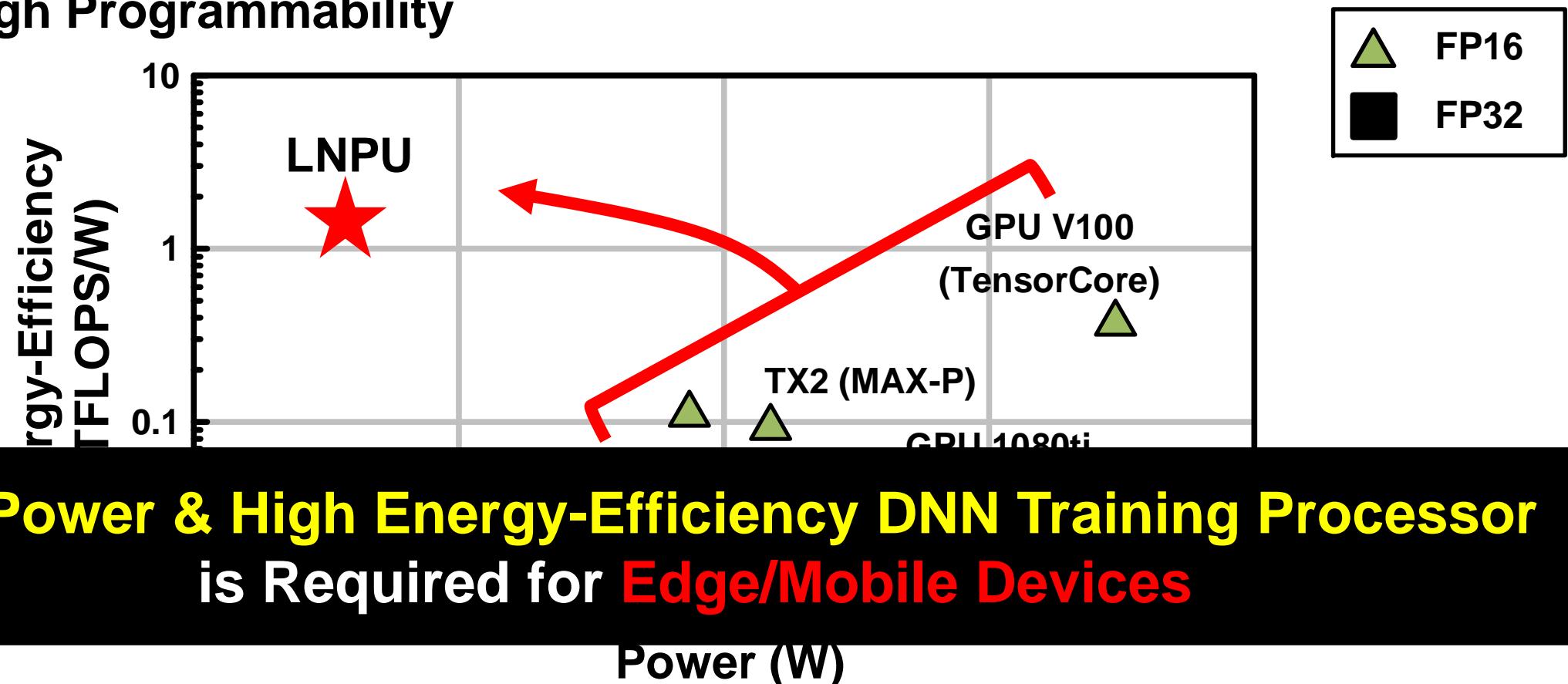


7.7: LNPU: A 25.3 TFLOPS/W Sparse Deep-Neural-Network Learning Processor with Fine-Grained Mixed Precision of FP8-FP16

DNN Training HW

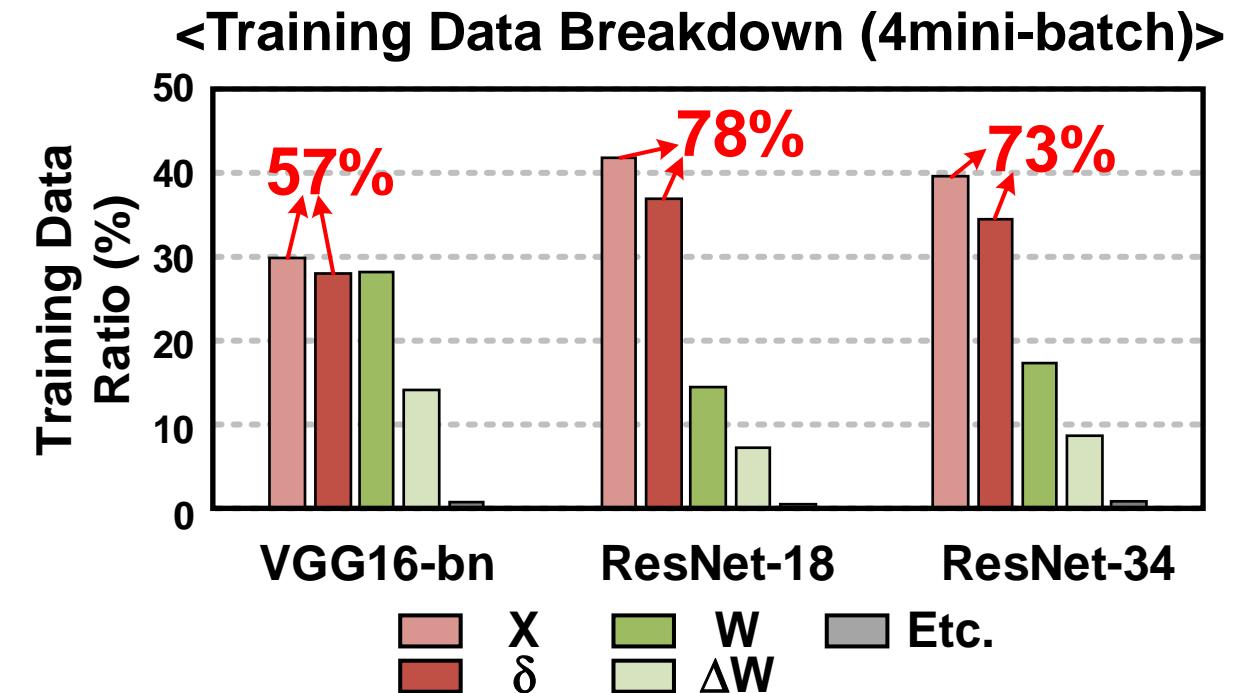
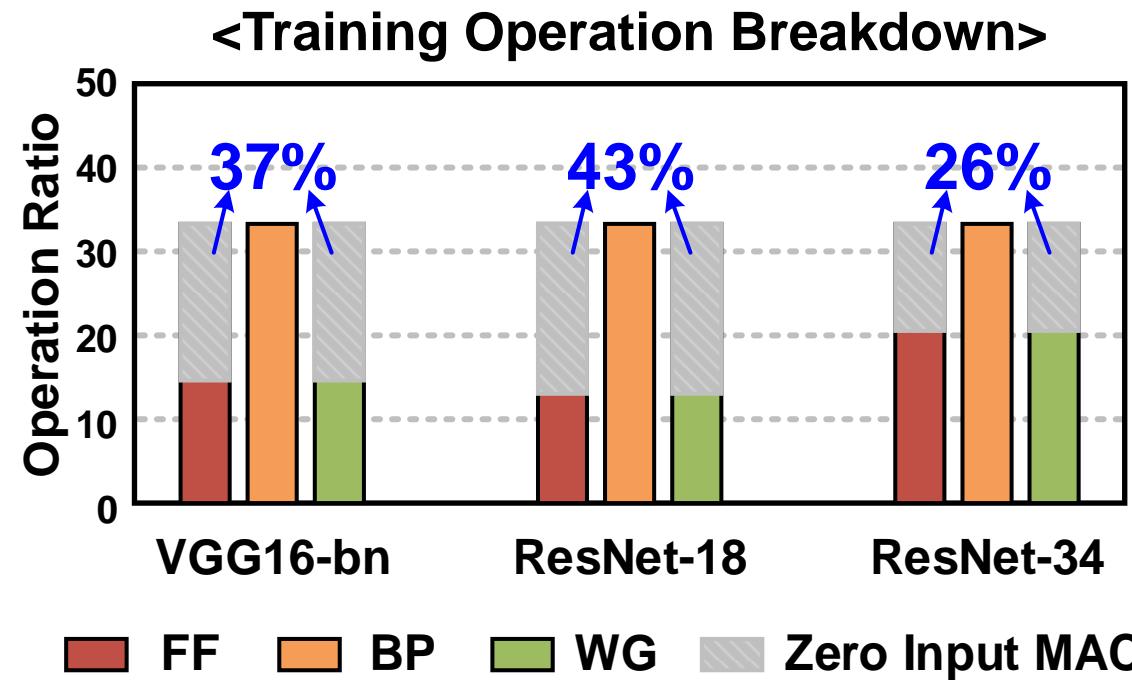
GPU: High Power Consumption & Low Energy-Efficiency

- Floating Point ALU
- High Programmability



Analysis of DNN Training

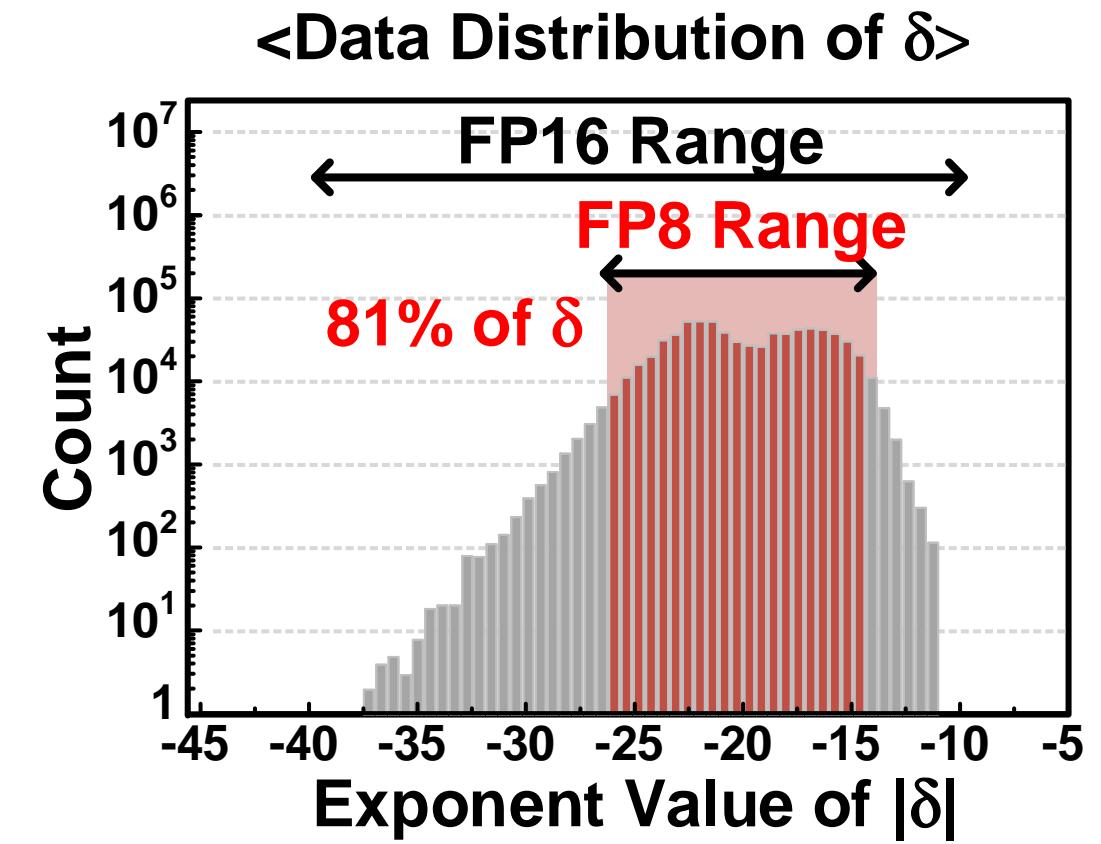
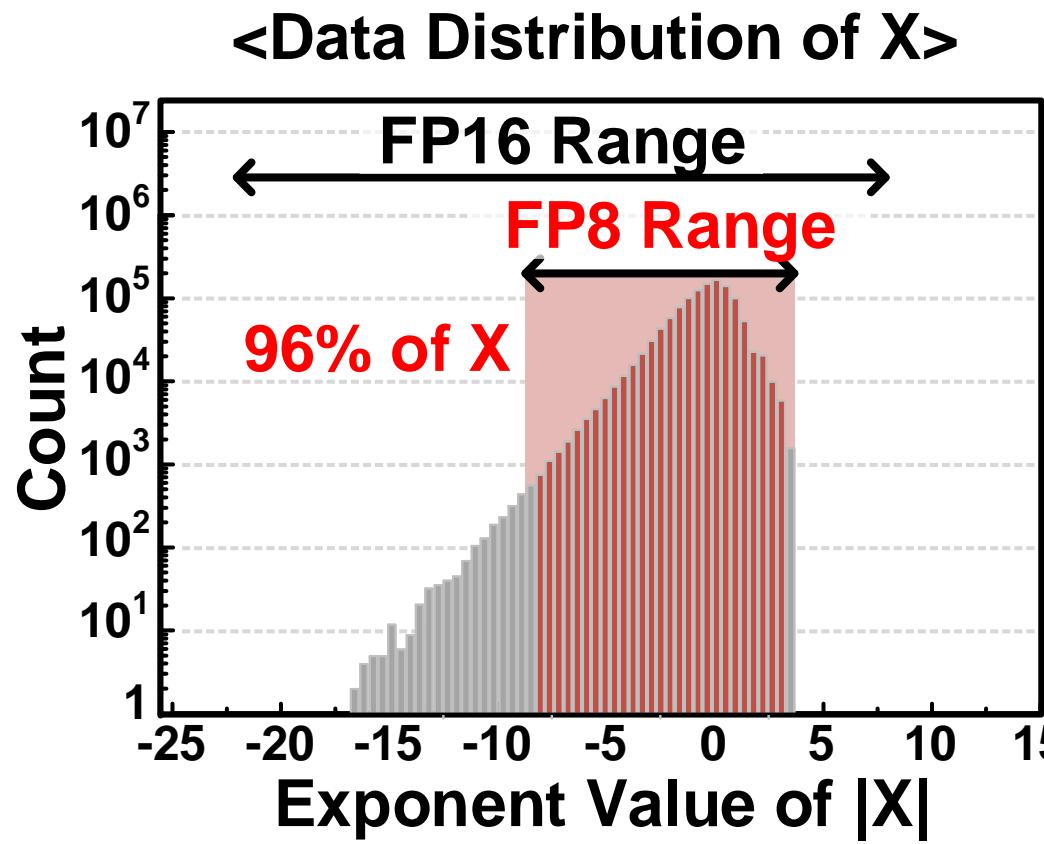
1. ~43% of Operation is Zero Input MAC Operation
2. X and δ occupy ~78% of External Memory Access (EMA)



Analysis of DNN Training

3. Data Distribution of X and δ

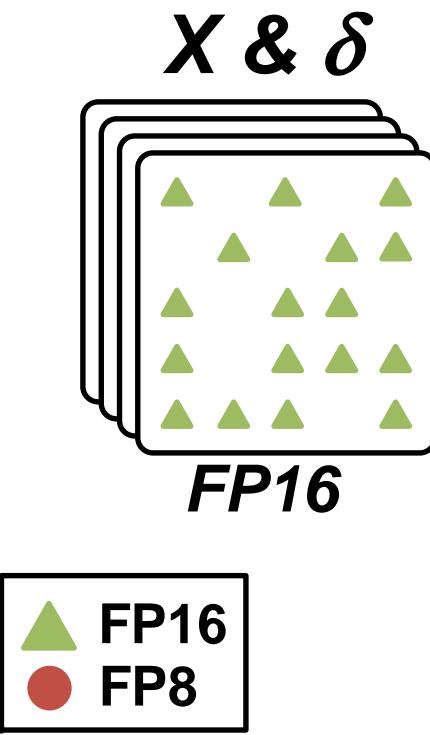
- FP8 → Cover ~80% of overall data @ ResNet-18 Conv4_1a



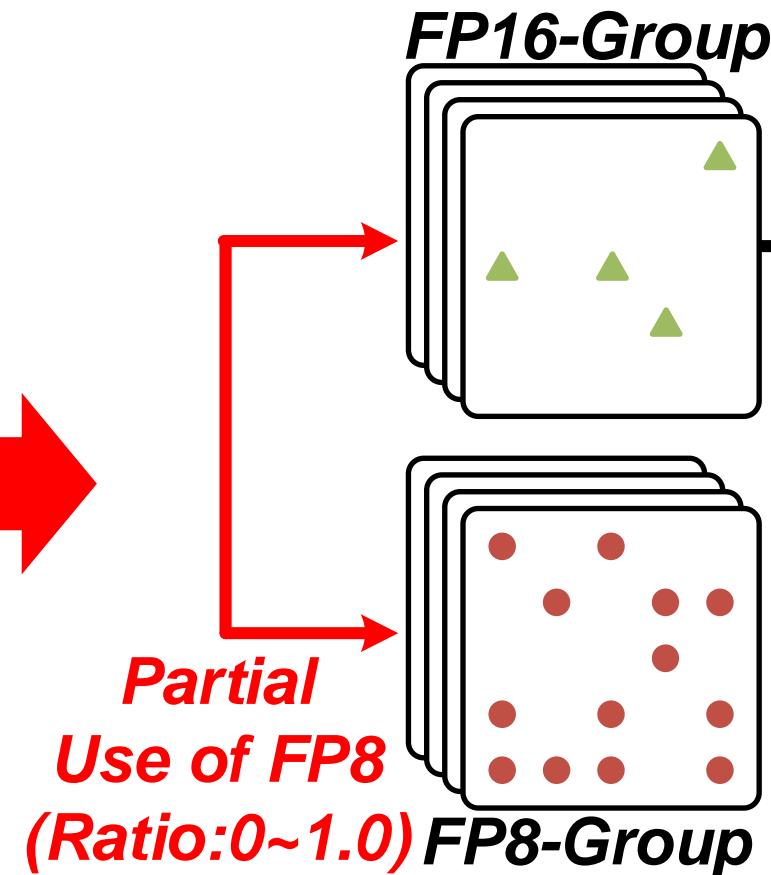
w/ Exp. Bias Shifting

Proposed Fine-grained Mixed Precision

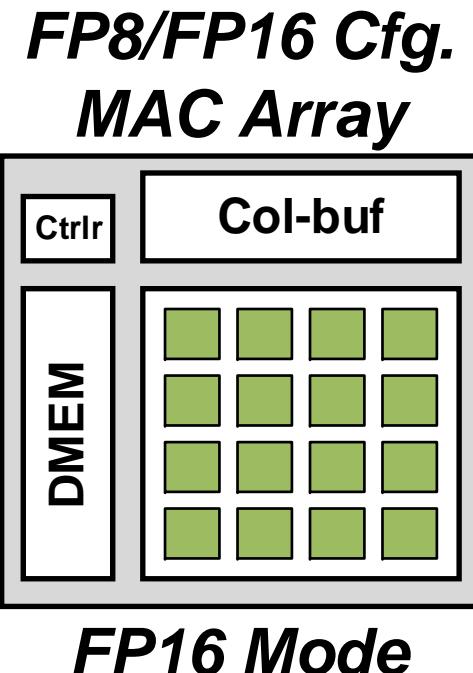
Conventional
(ex. GPU ..)



Fine-grained Mixed Precision (FGMP)

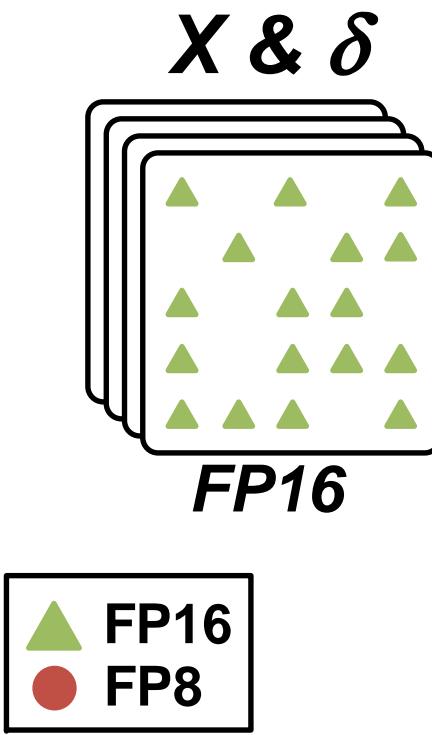


High Precision

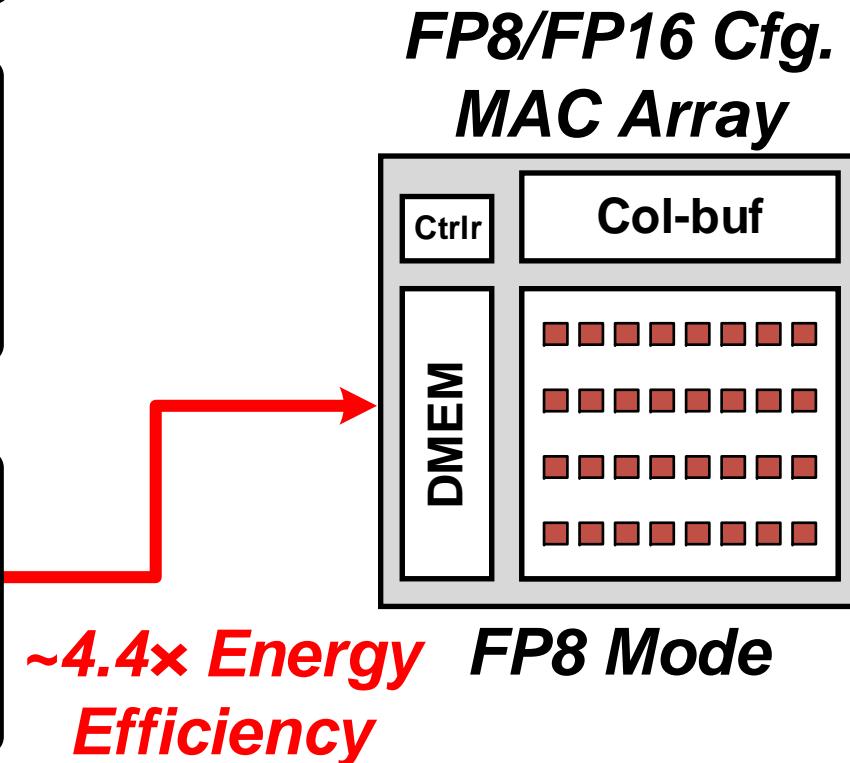
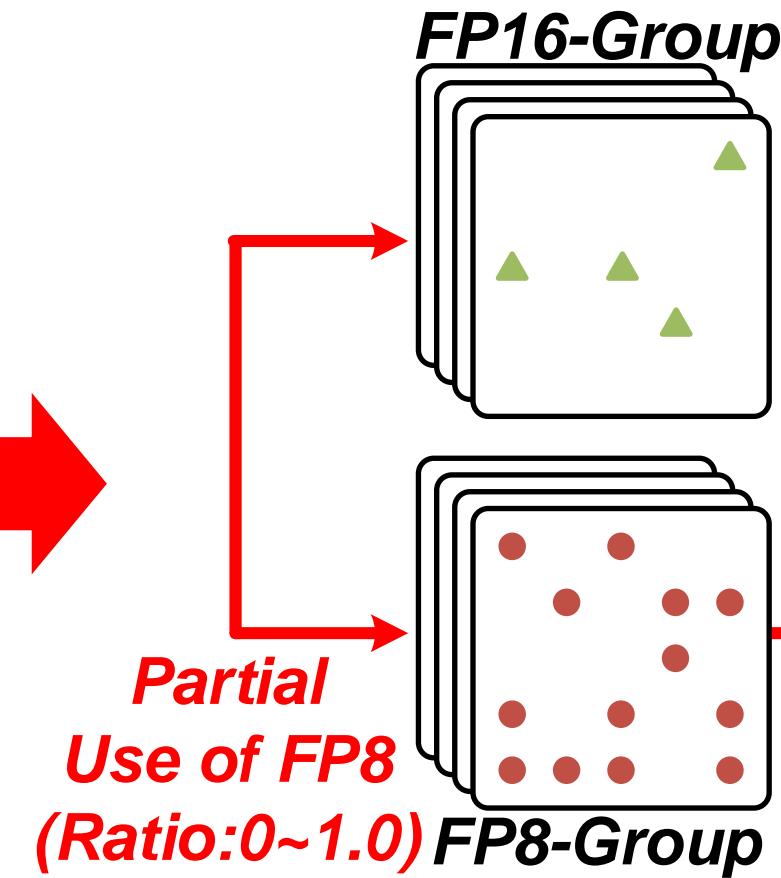


Proposed Fine-grained Mixed Precision

Conventional
(ex. GPU ..)

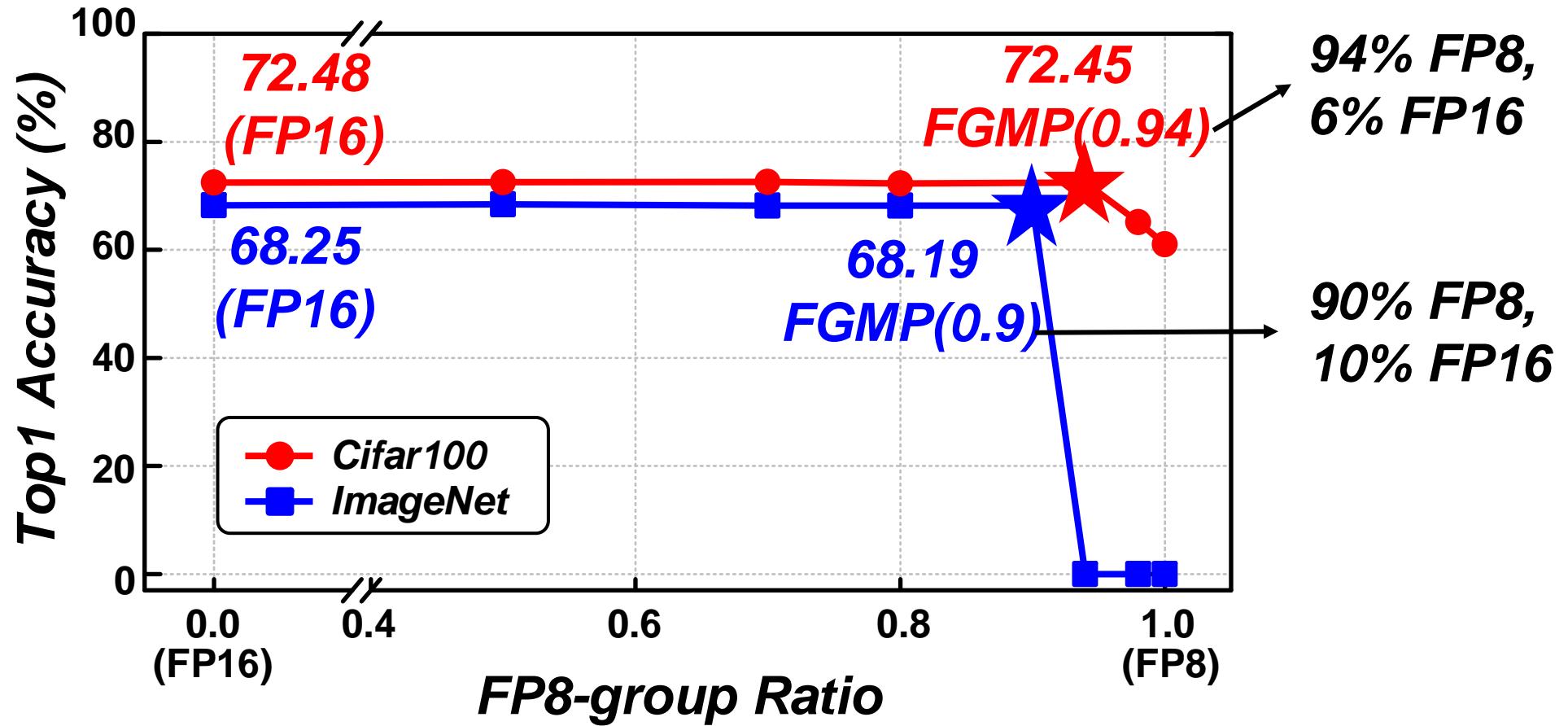


Fine-grained Mixed Precision (FGMP)



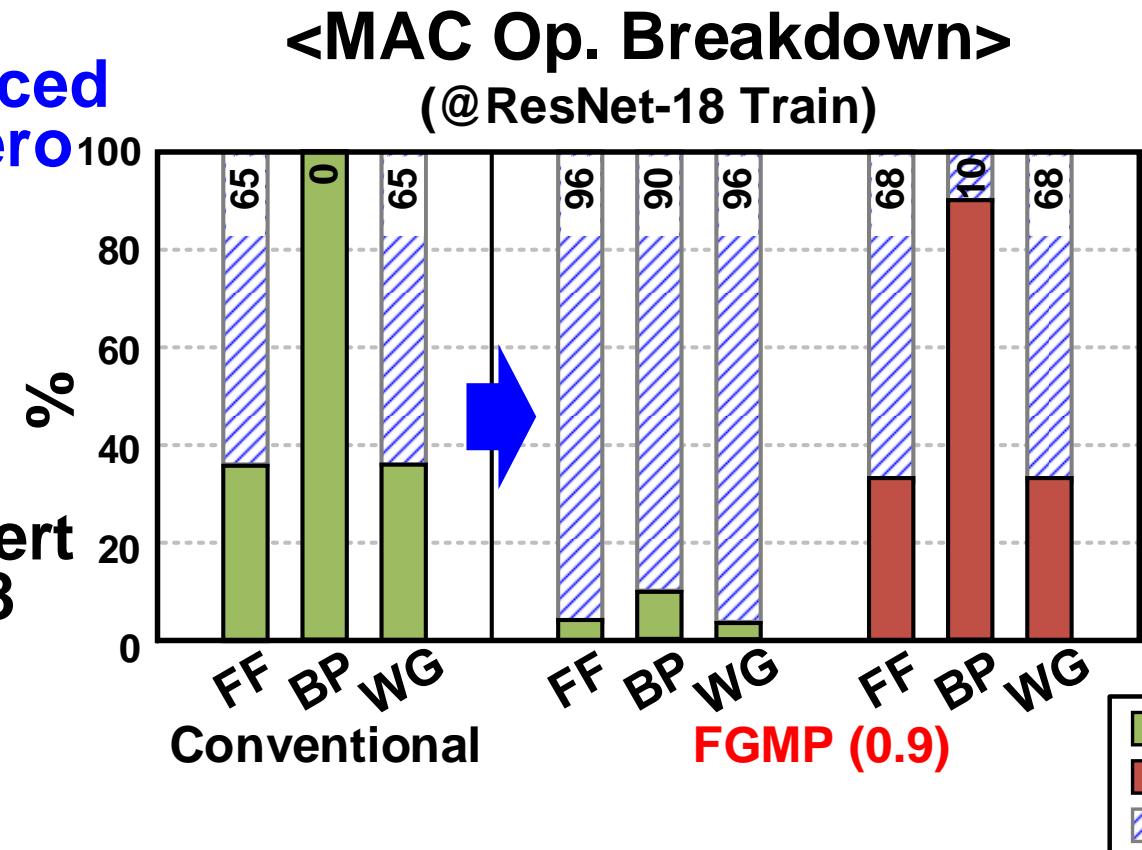
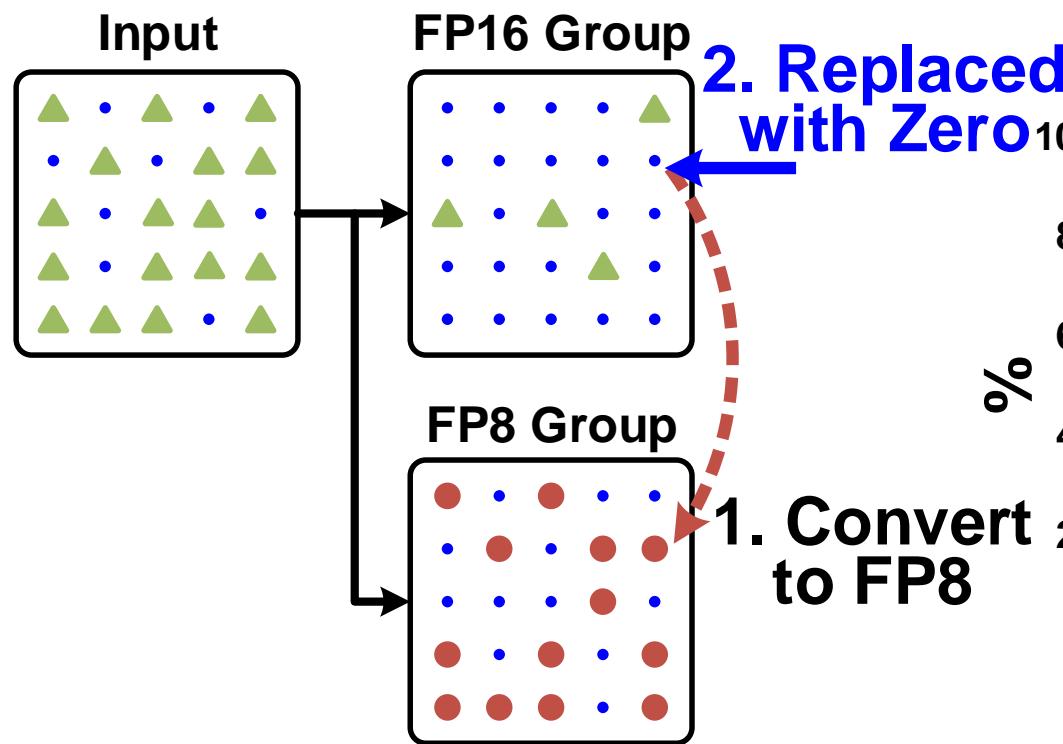
Proposed Fine-grained Mixed Precision

□ ResNet-18 Training Results with FGMP



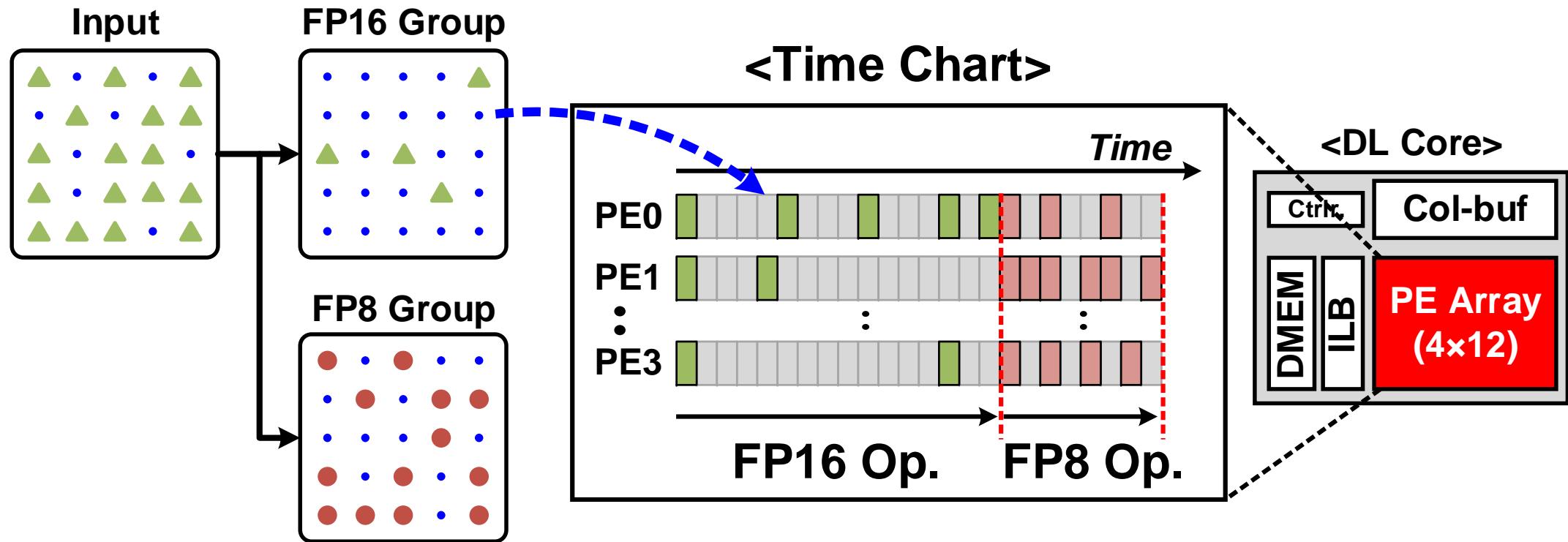
Zero Input MAC Op. by FGMP

- Increase Zeros Input MAC Op. for All Training Steps



Zero Input MAC Op. by FGMP

- Require Zero-Skipping Op. & Data-path for FP8/FP16



Proposed Processor: LNPU

1. Fully Reconfigurable Sparse DL Core – High Throughput

- ✓ Support Zero-Skipping Operation for All DNN Training Steps

2. Input Load Balancer – High PE Utilization

- ✓ Balance Input between PEs

3. FP8/FP16 Configurable PE – High Energy Efficiency

- ✓ Accelerate FGMP Encoded Operand

A 25.3 TFLOPS/W DNN Learning Processor
Supporting Zero-Skipping for All Training Steps
with Fine-Grained Mixed Precision of FP8-FP16

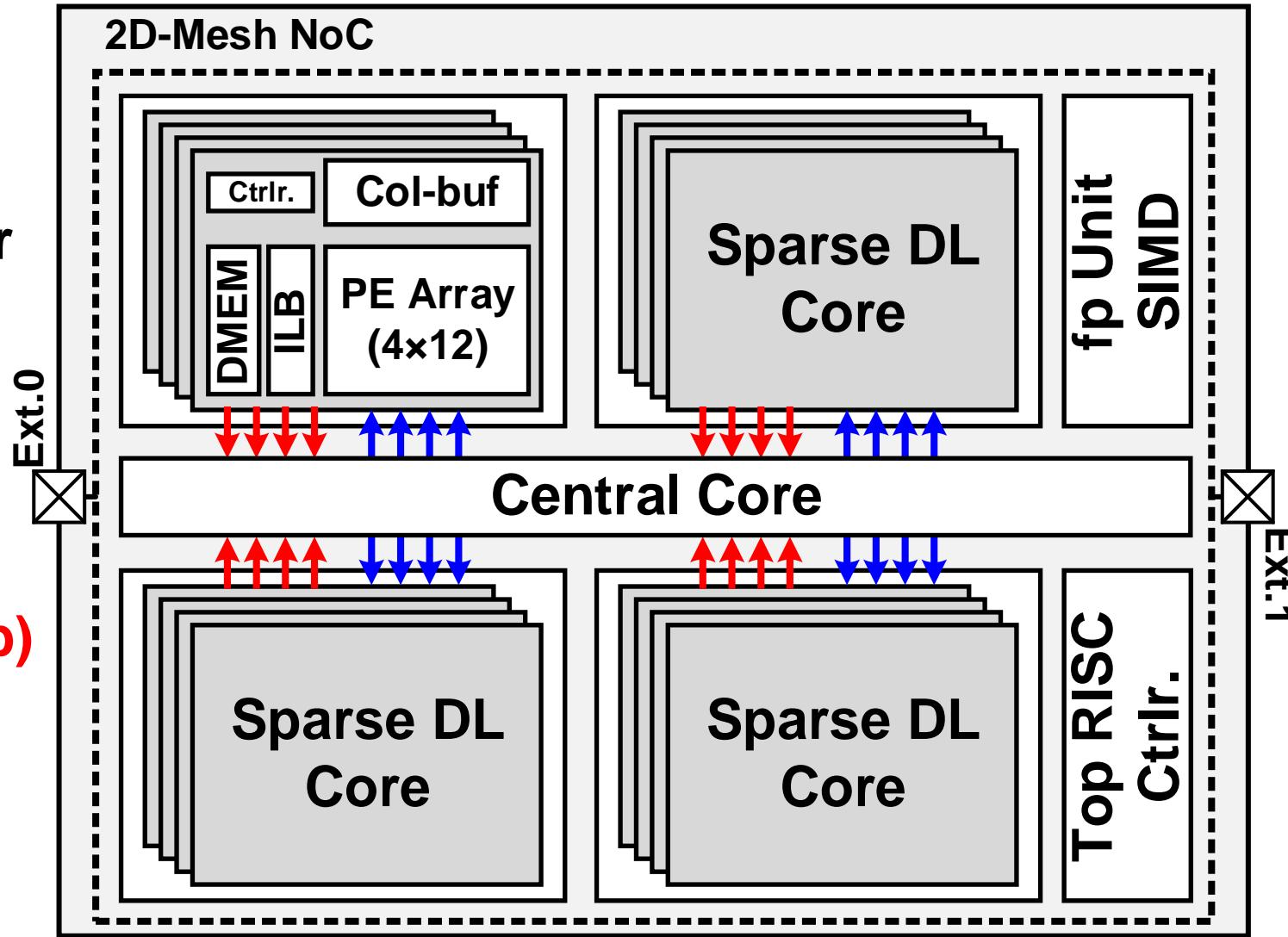
Overall Architecture of LNPU

Sparse DL Core

1. Support FGMP
2. Support Zero-Skipping for All DNN Training Steps

Central Core

1. Aggregate Output P_{SUM} of Cores (@FF Step, BP Step)
2. Feed Error to each Core (@WG Step)



Architecture of Sparse DL Core

□ Local Router

- Support Zero-Skipping MAC Op.

→ Improve Throughput

□ Input Load Balancer (ILB)

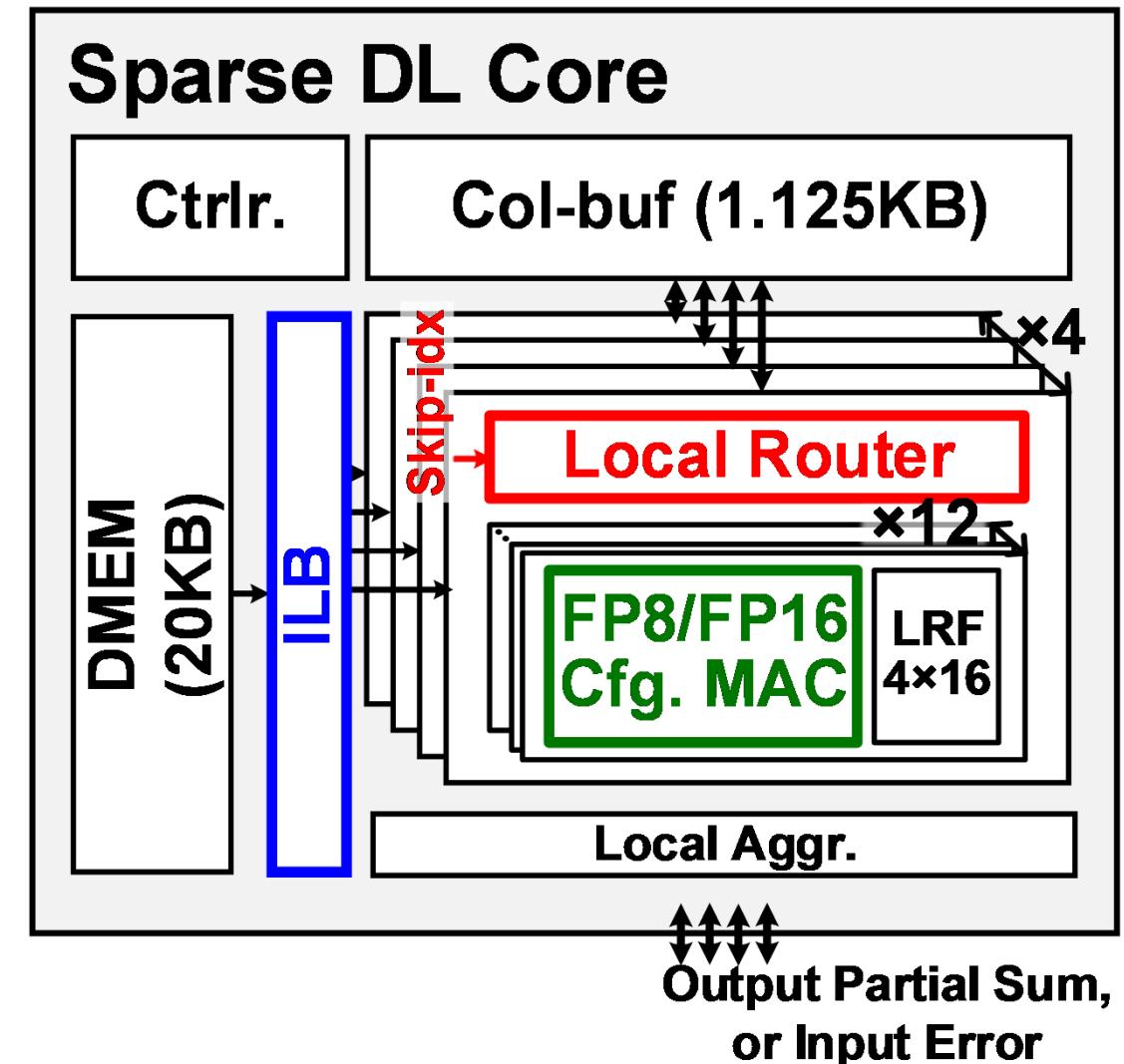
- Balance Input btw. PEs

→ Improve PE Utilization

□ FP8/FP16 Configurable PE

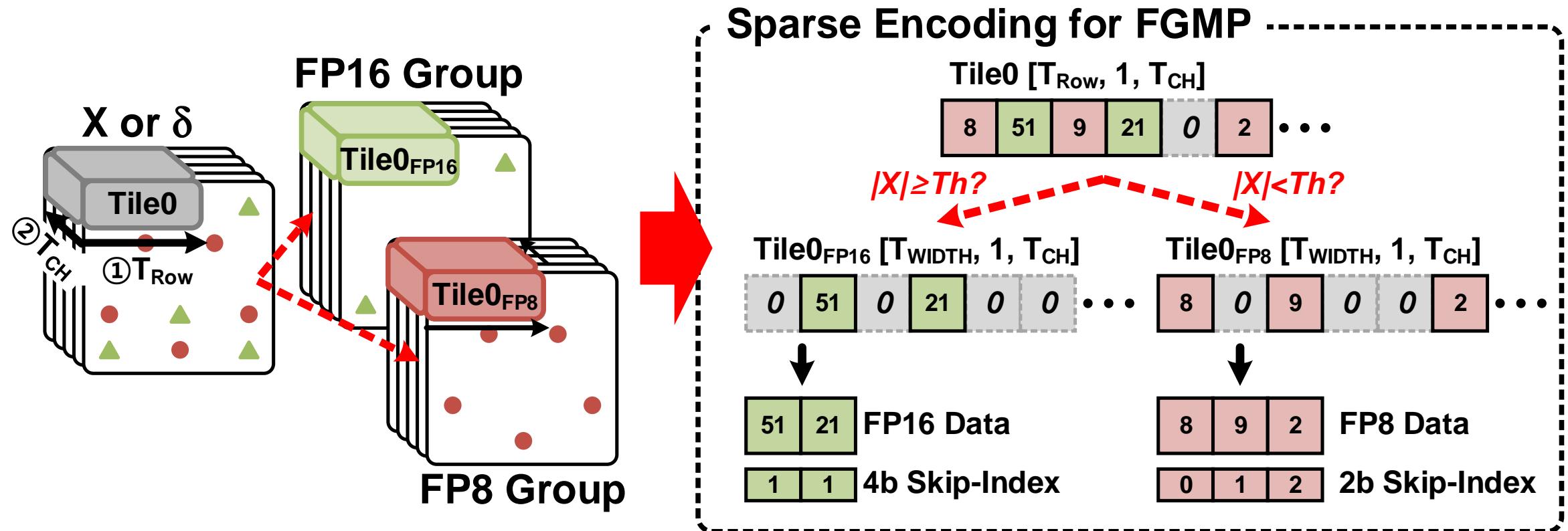
- Support FP8/FP16 Data-path

→ Enhance Energy-Efficiency



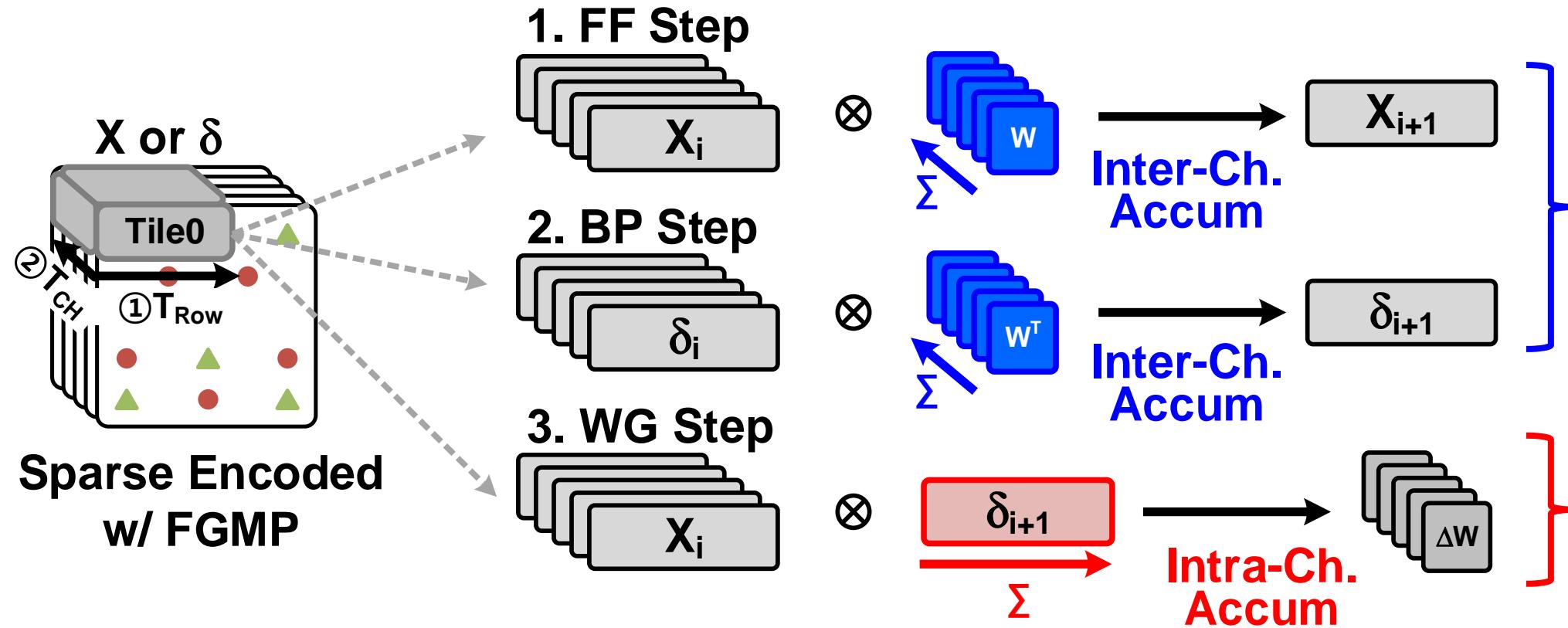
Sparse Encoding for FGMP

- ❑ X or $\delta \rightarrow$ Divided into FP8/FP16 Groups
- ❑ Zero-Run-Length Encoding for each Group



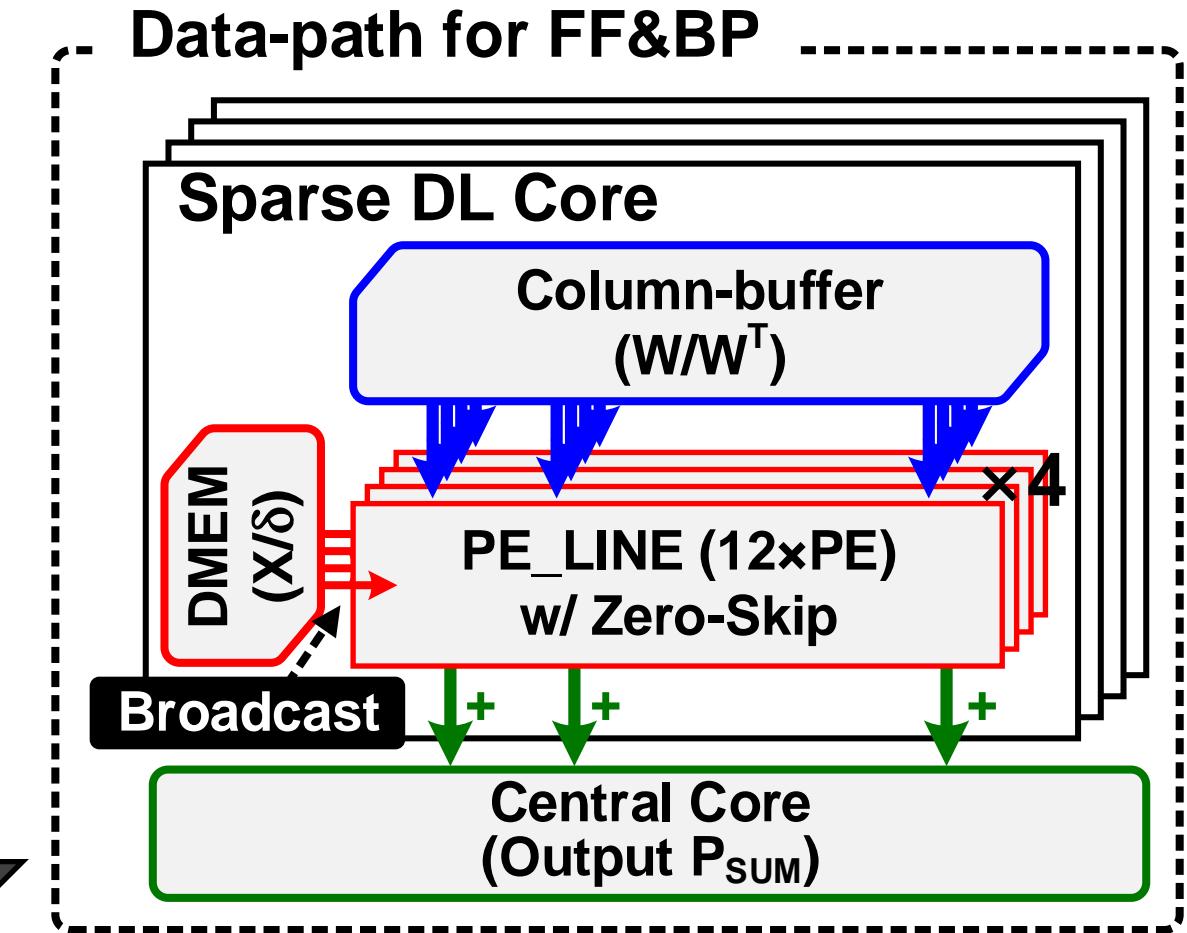
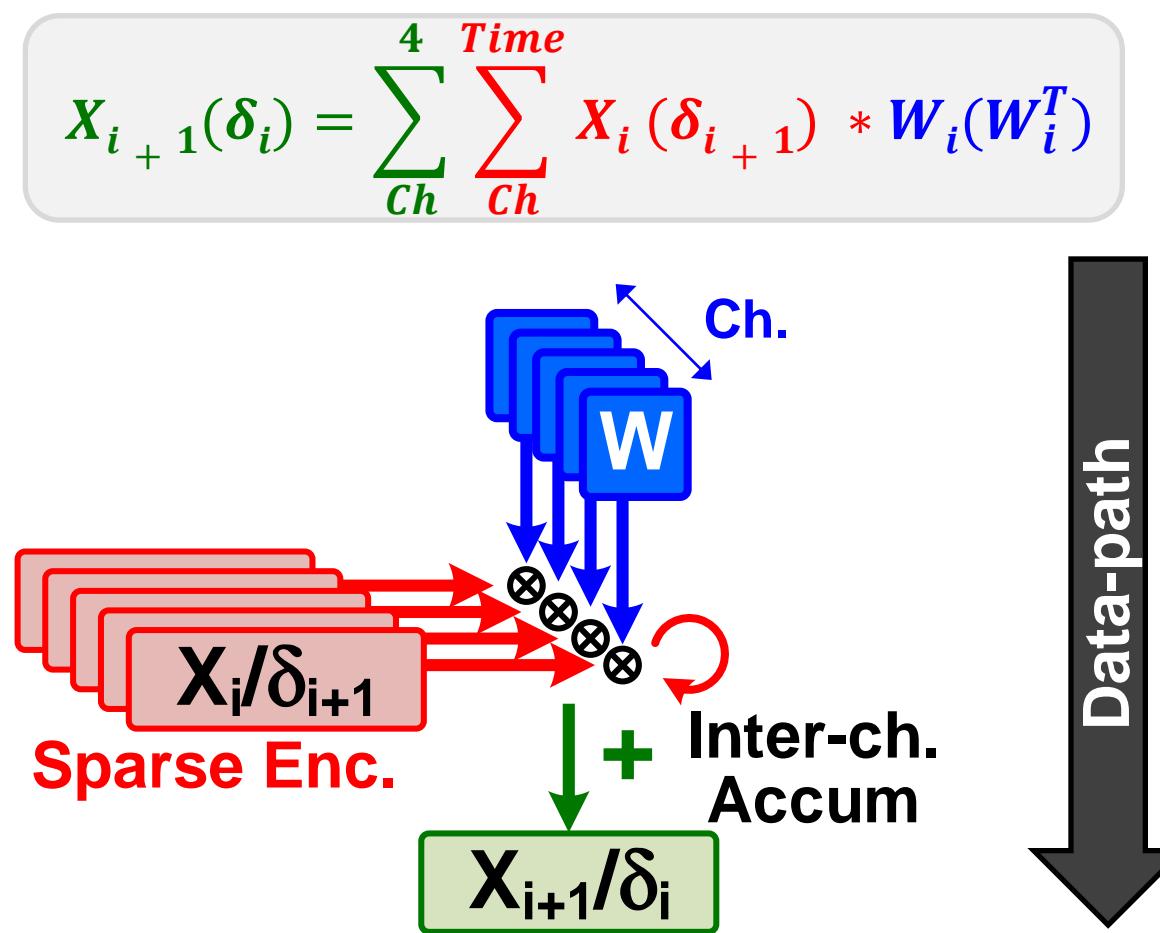
Data-path for Sparse DNN Training

- Inter-Channel Accumulation Path for FF & BP Steps
- Intra-Channel Accumulation Path for WG Step



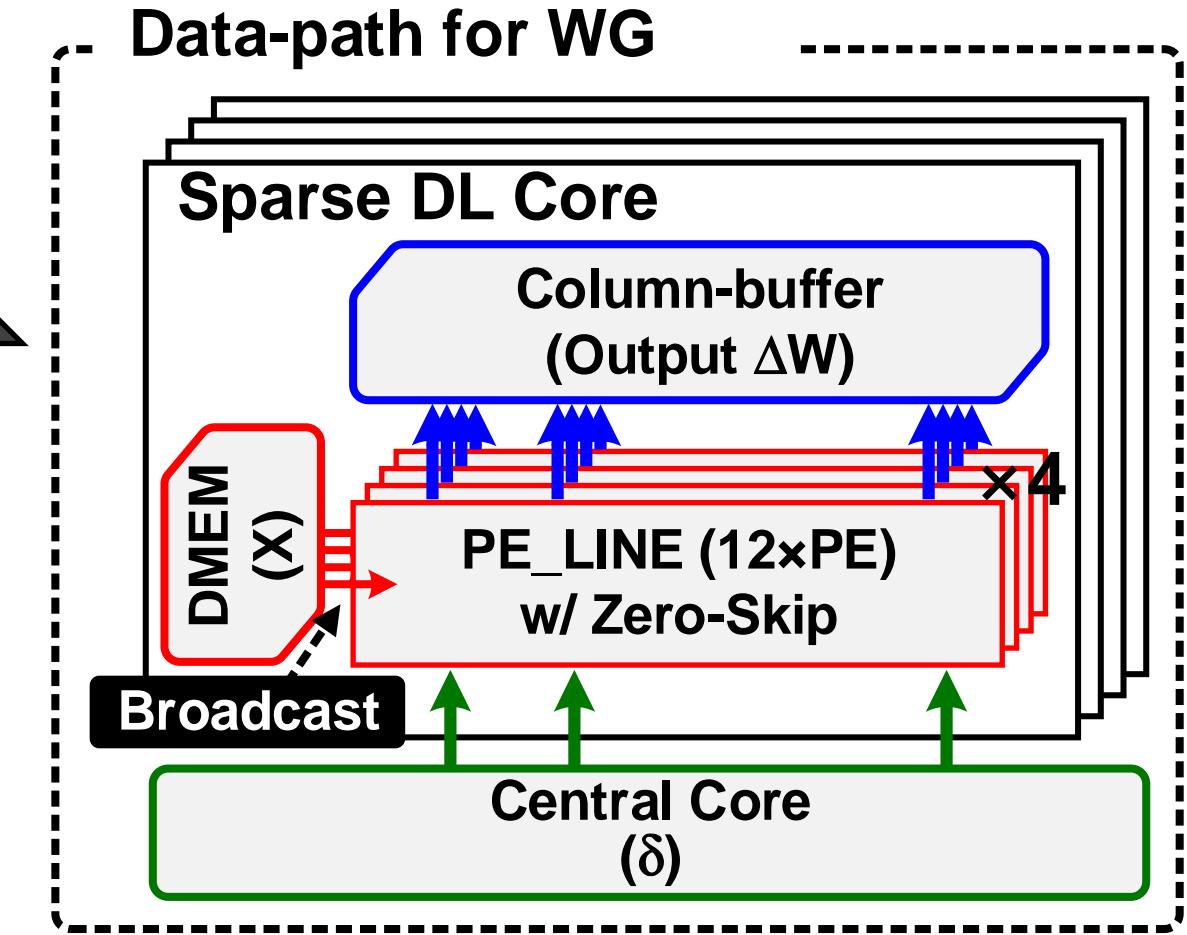
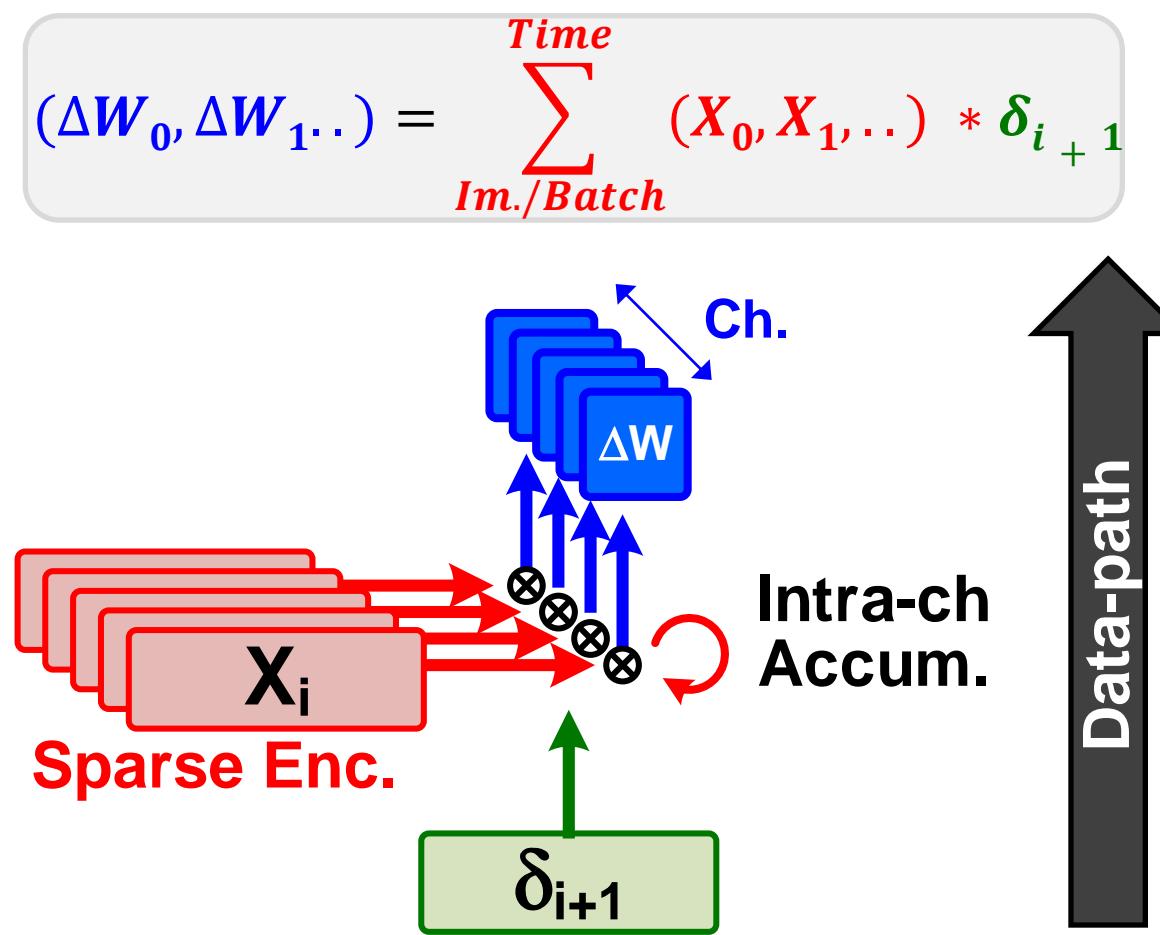
Fully Reconfigurable Sparse DL Core

□ Data-path for FF & BP Steps with Sparse Encoded Input



Fully Reconfigurable Sparse DL Core

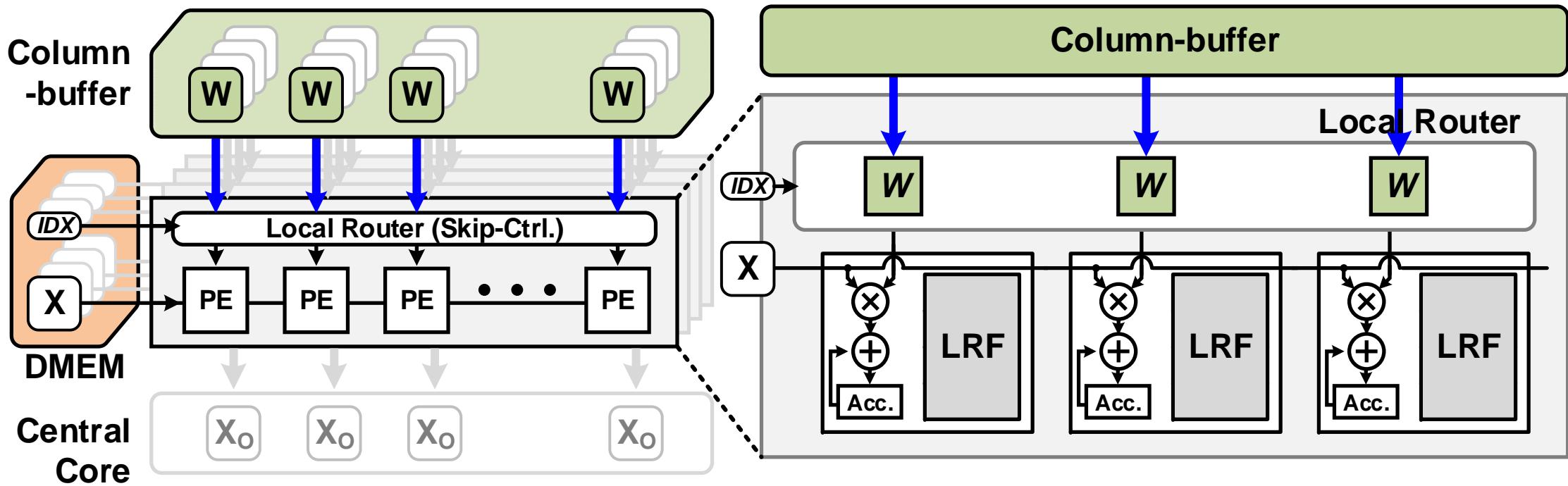
□ Data-path for WG Step with Sparse Encoded Input



Zero Skipping Operation (FF & BP)

□ FF & BP Steps

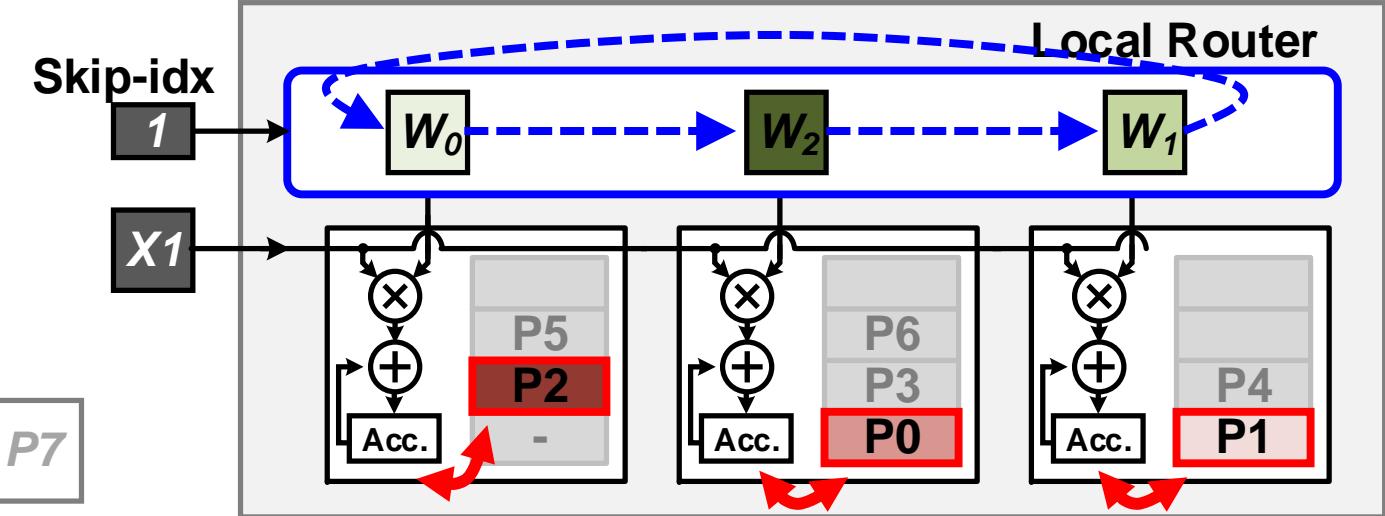
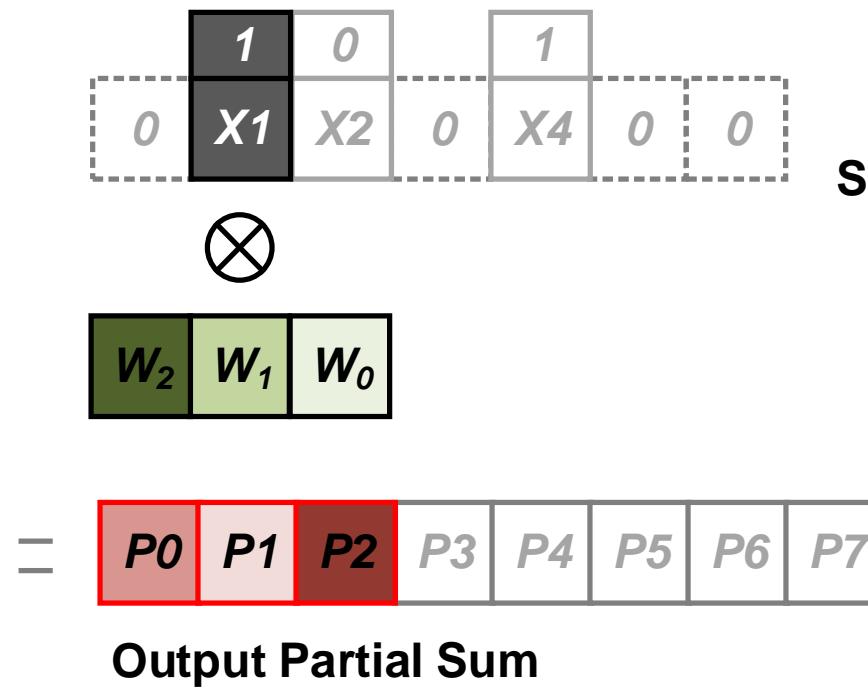
- $W / W^T \rightarrow$ Feed to Local Router from Col-buf.
- Sparse Encoded $X / \delta \rightarrow$ Broadcasted to PE_LINE (Row of PE array)



Zero Skipping Operation (FF & BP)

□ FF & BP Steps (for 3×3 Weight, @T0 Cycle)

- Local Router → Shift W / W^T controlled by skip-index
- LRF → Store output partial sum

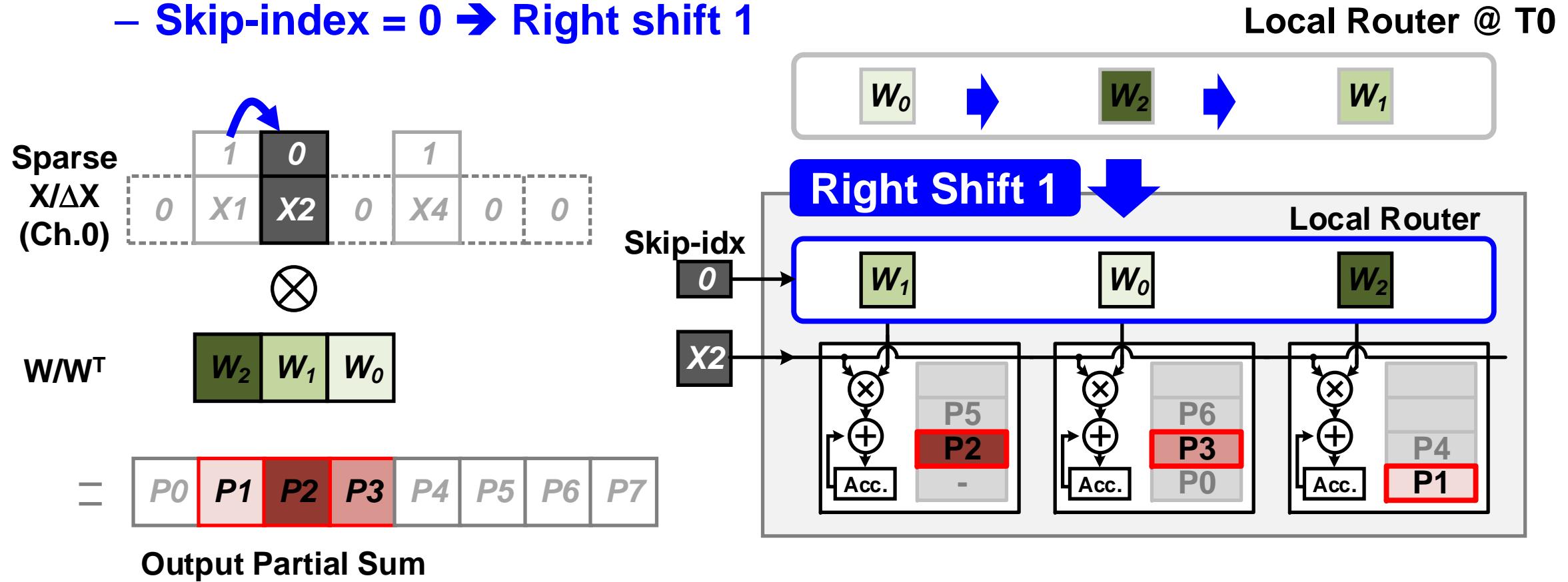


Output Partial Sum

Zero Skipping Operation (FF & BP)

□ FF & BP Steps (for 3×3 Weight, @T1 Cycle)

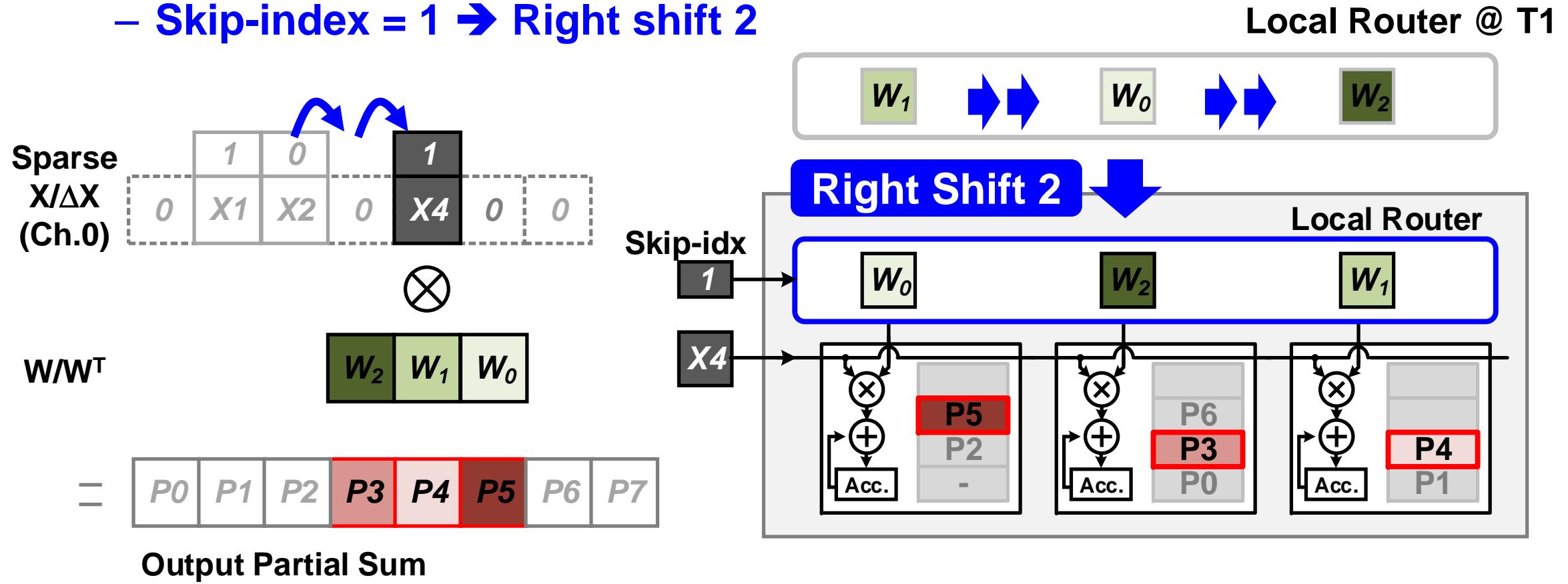
- Local Router → Shift W / W^T controlled by skip-index
- Skip-index = 0 → Right shift 1



Zero Skipping Operation (FF & BP)

□ FF & BP Steps (for 3×3 Weight, @T2 Cycle)

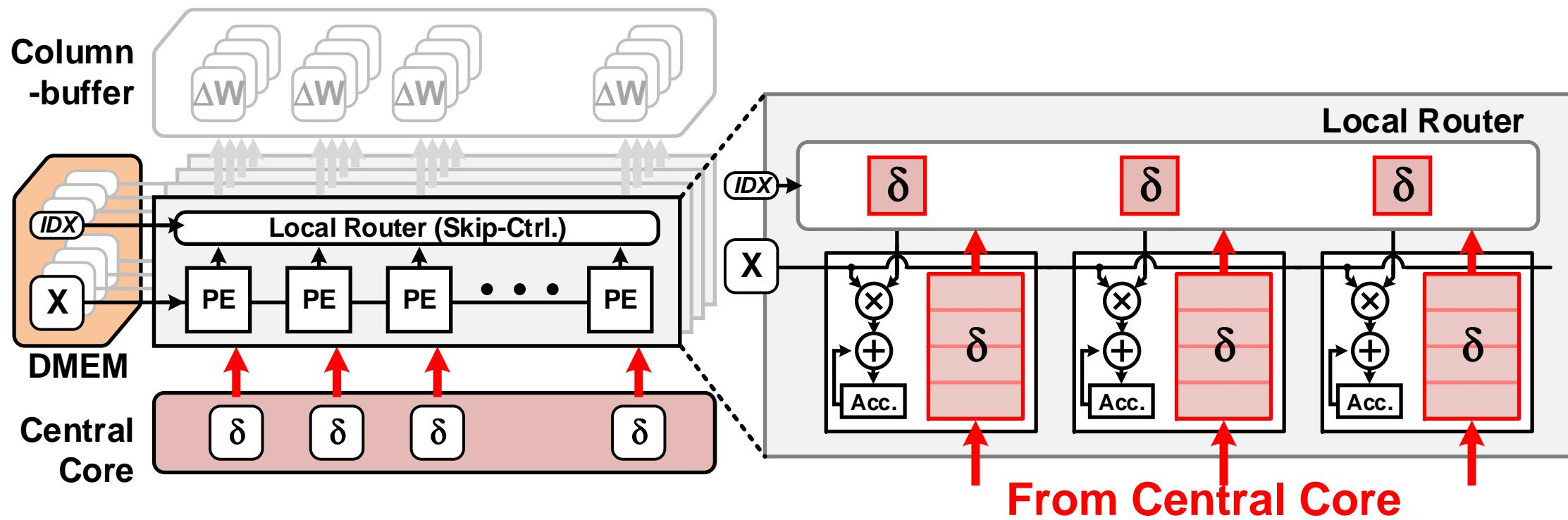
- Local Router → Shift W / W^T controlled by skip-index
- Skip-index = 1 → Right shift 2



Zero Skipping Operation (WG)

WG Step (for 3×3 Weight Gradient)

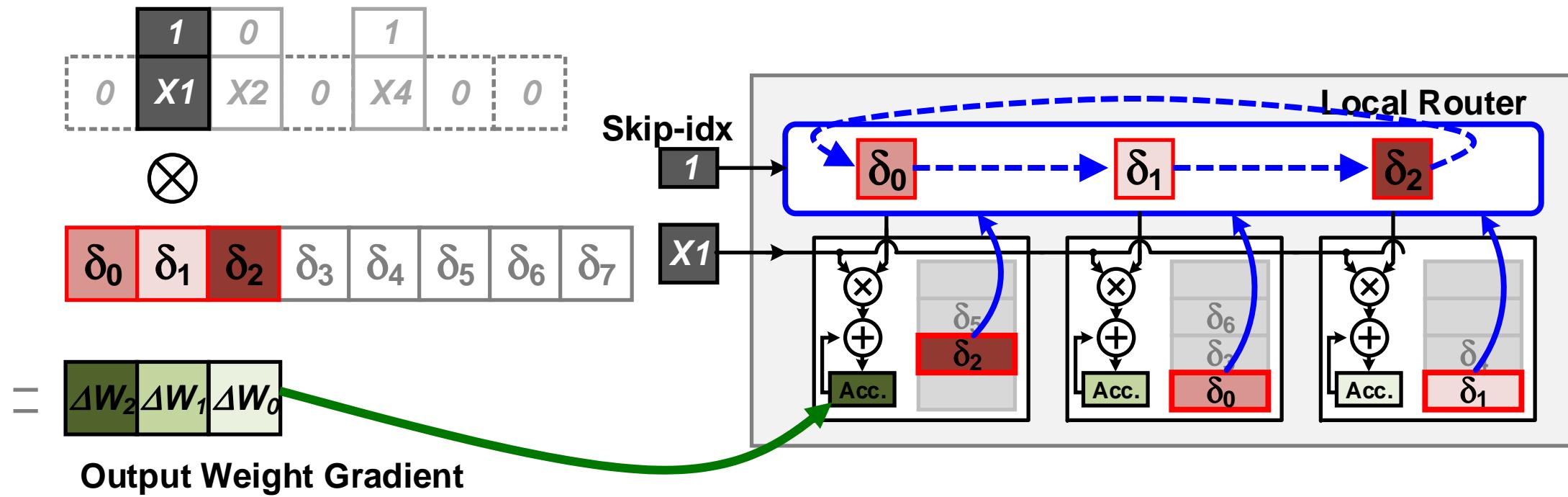
- δ : Loaded from Central Core & Stored on LRF
- Sparse Encoded X → Broadcasted to PE_LINE



Zero Skipping Operation (WG)

WG Step (for 3×3 Weight Gradient @ T0 Cycle)

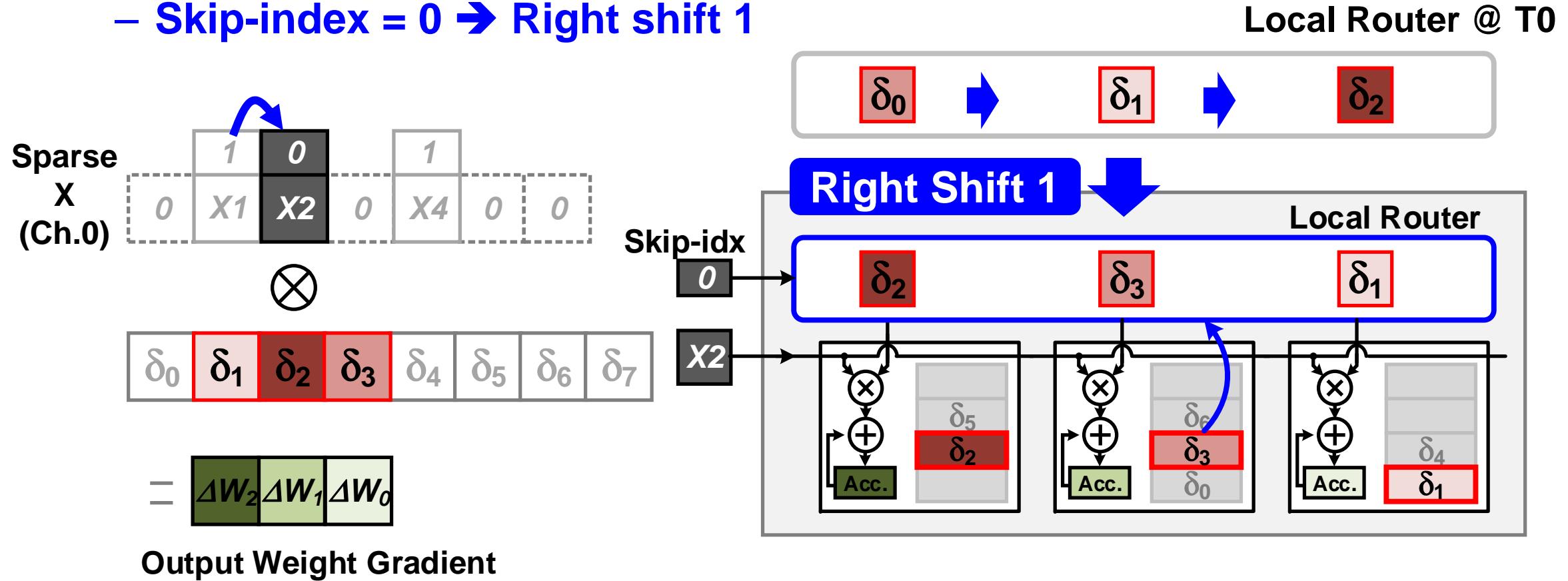
- LRF → Feed δ to Local Router
- Local Router → Shift δ controlled by skip-index
- MAC Accum. Reg → Accumulate ΔW



Zero Skipping Operation (WG)

WG Step (for 3×3 Weight Gradient @ T1 Cycle)

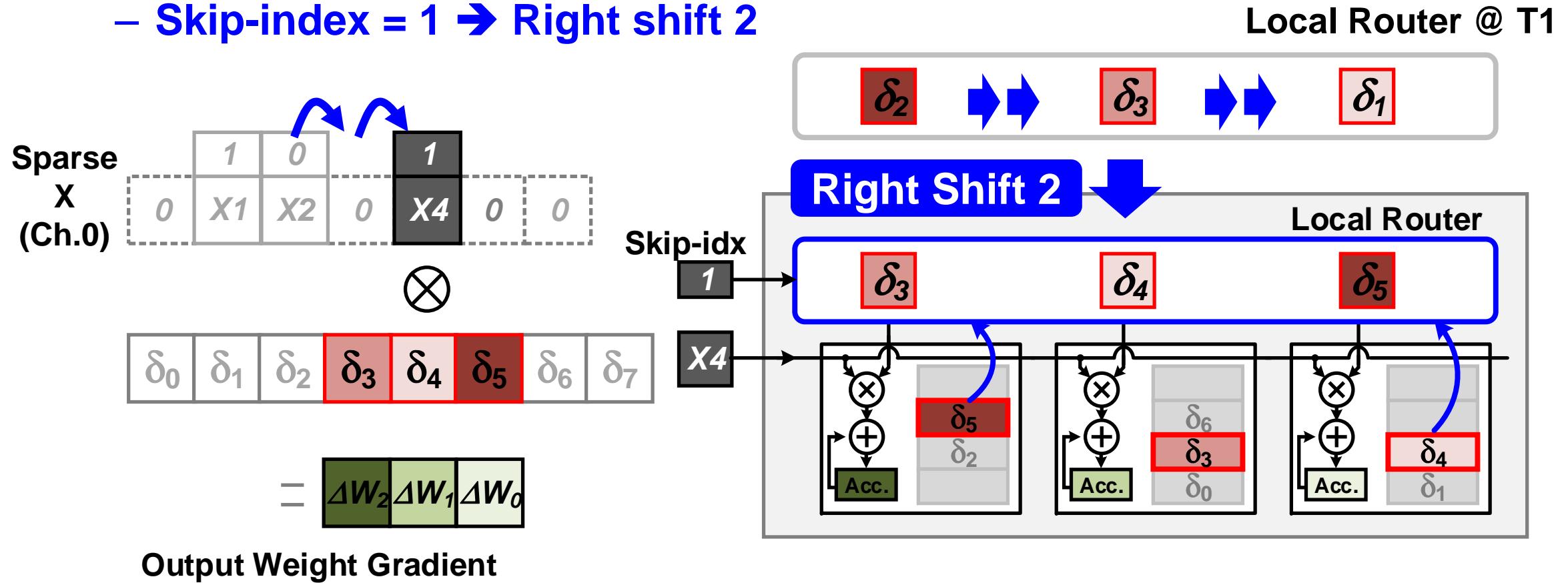
- Local Router → Shift δ controlled by skip-index
- Skip-index = 0 → Right shift 1



Zero Skipping Operation (WG)

WG Step (for 3×3 Weight Gradient @ T2 Cycle)

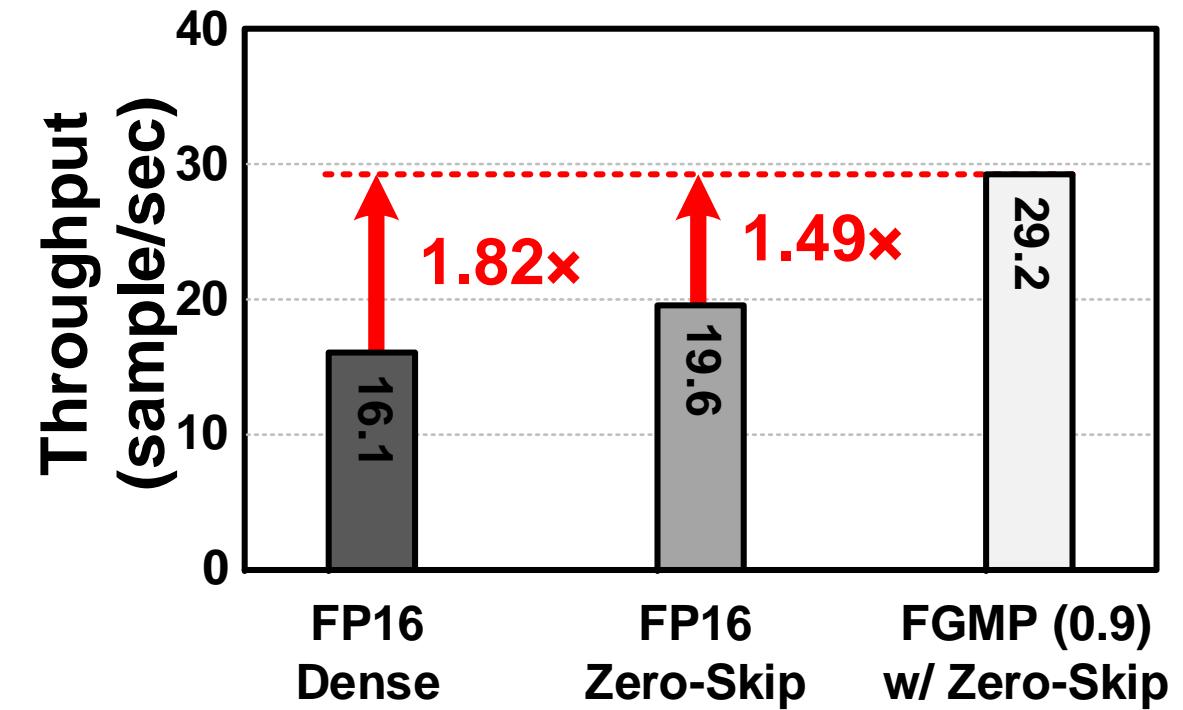
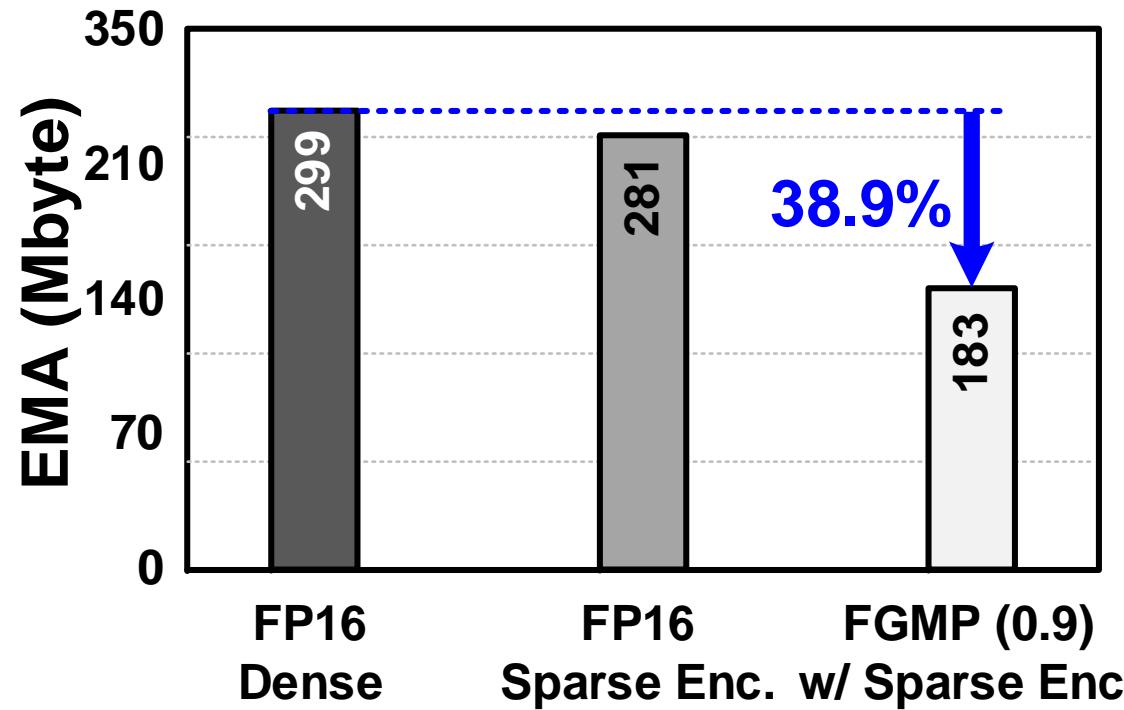
- Local Router → Shift δ controlled by skip-index
- Skip-index = 1 → Right shift 2



Fully Reconfigurable Sparse DL Core

□ EMA & Performance for ResNet-18 Training

- 38.9% EMA reduction & 1.82× performance improvement

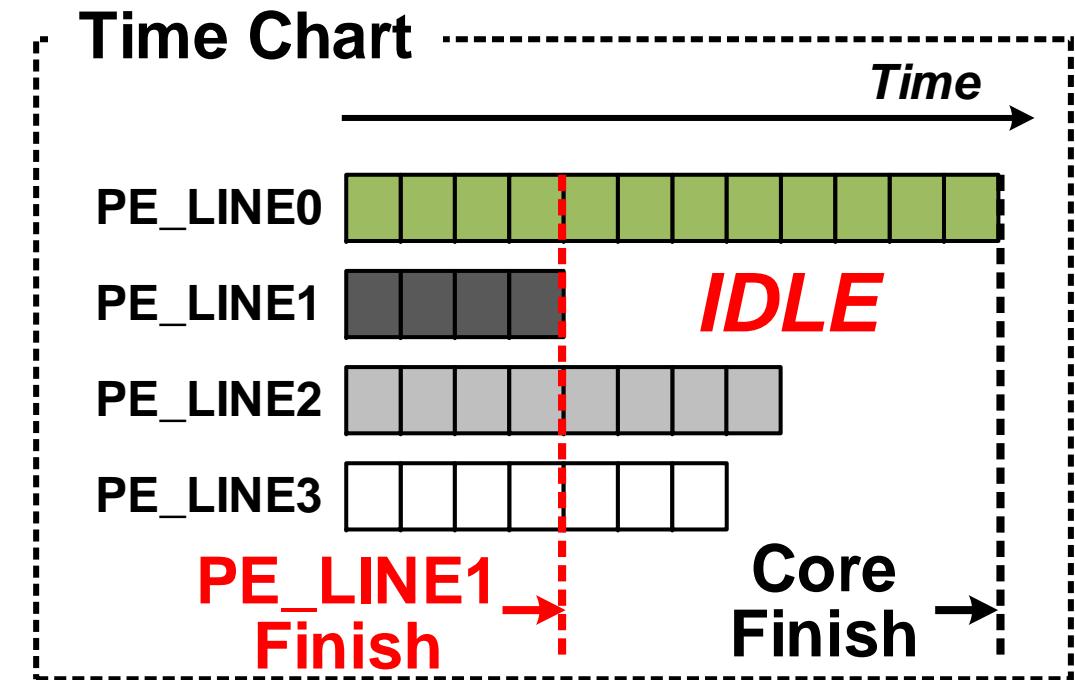
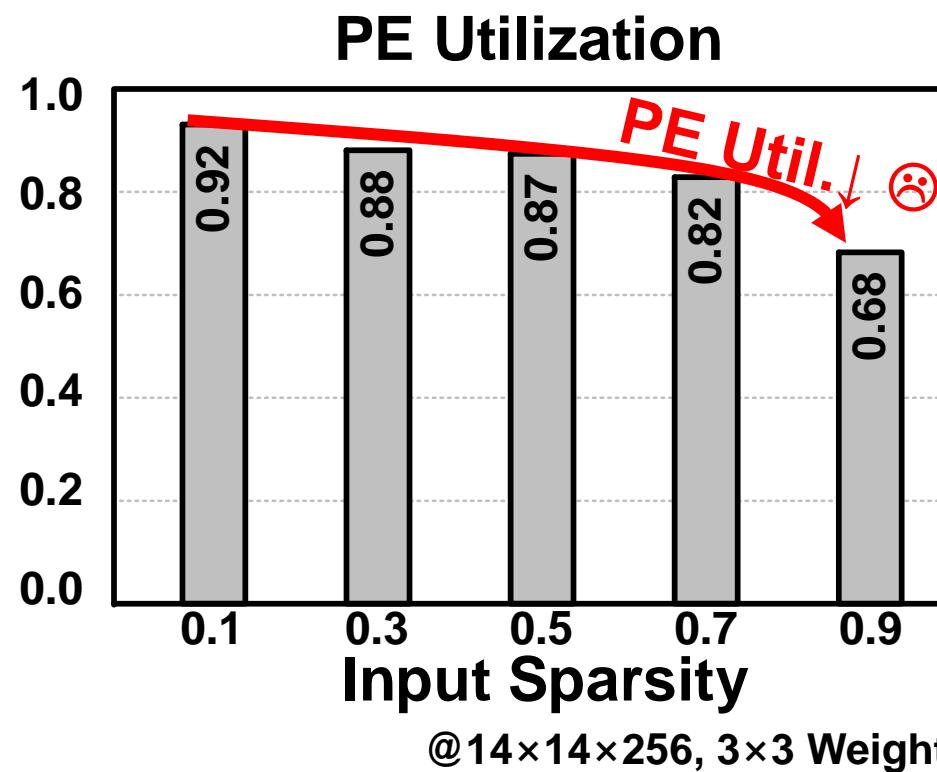


@ 4 Mini-Batch Size

Input Load Balancer (ILB)

Irregular Sparsity of Input Tiles

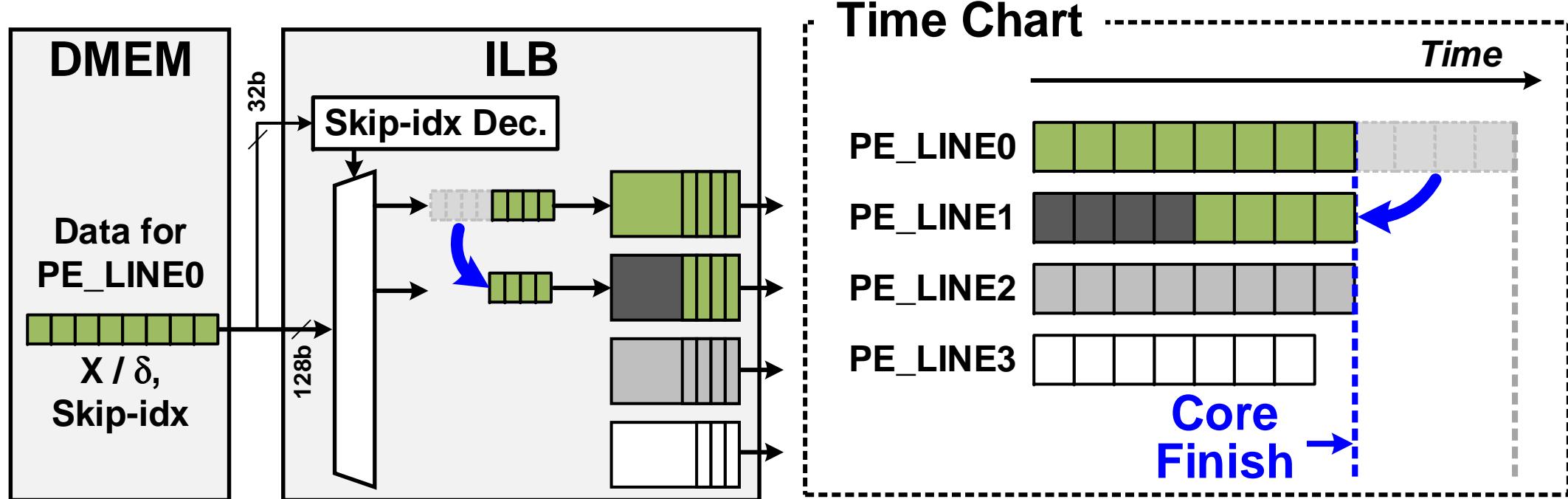
- Degrade PE utilization by imbalanced input btw. PE_LINES



Input Load Balancer (ILB)

□ Balance Input between PE_LINES

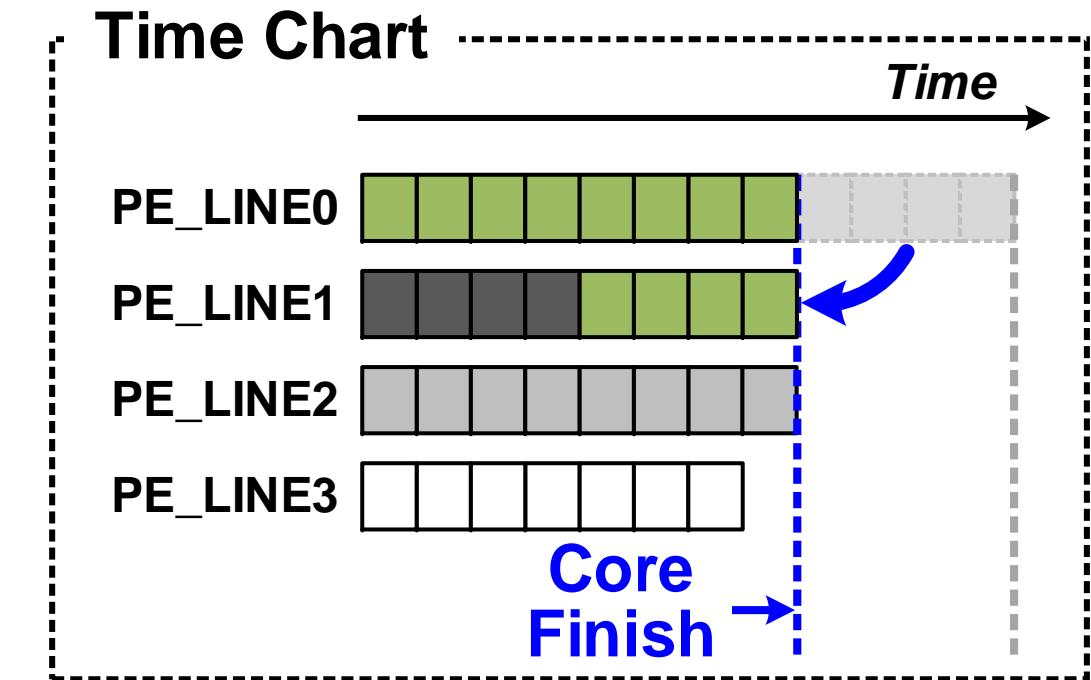
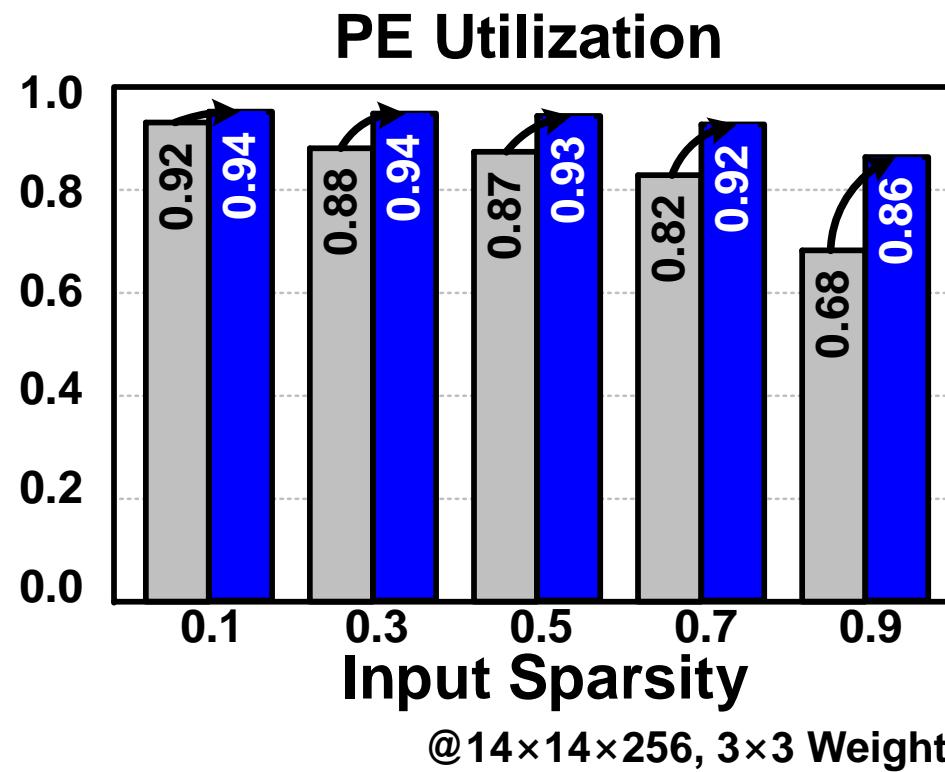
- Data for busy PE_LINE → Distribute to idle PE_LINE



Input Load Balancer (ILB)

❑ Balance Input between PE_LINES

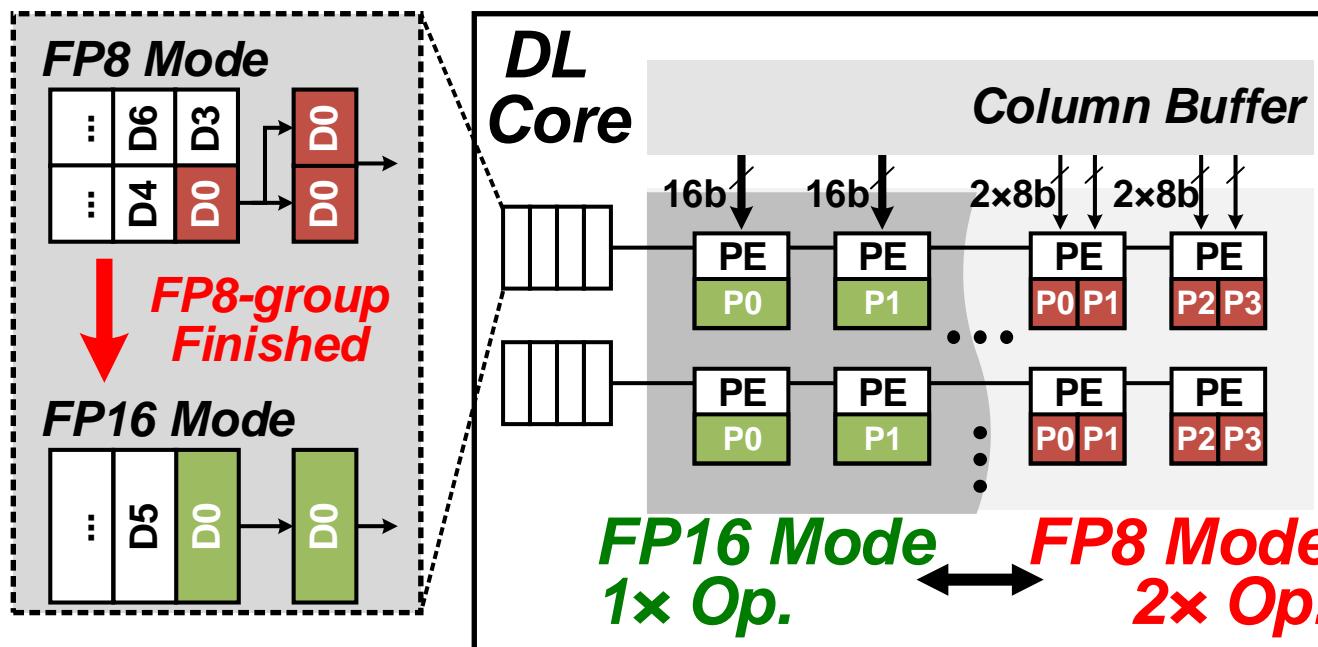
- Data for busy PE_LINE → Distribute to idle PE_LINE



FP8/FP16 Configurable PE

❑ Accelerate FGMP Encoded Input

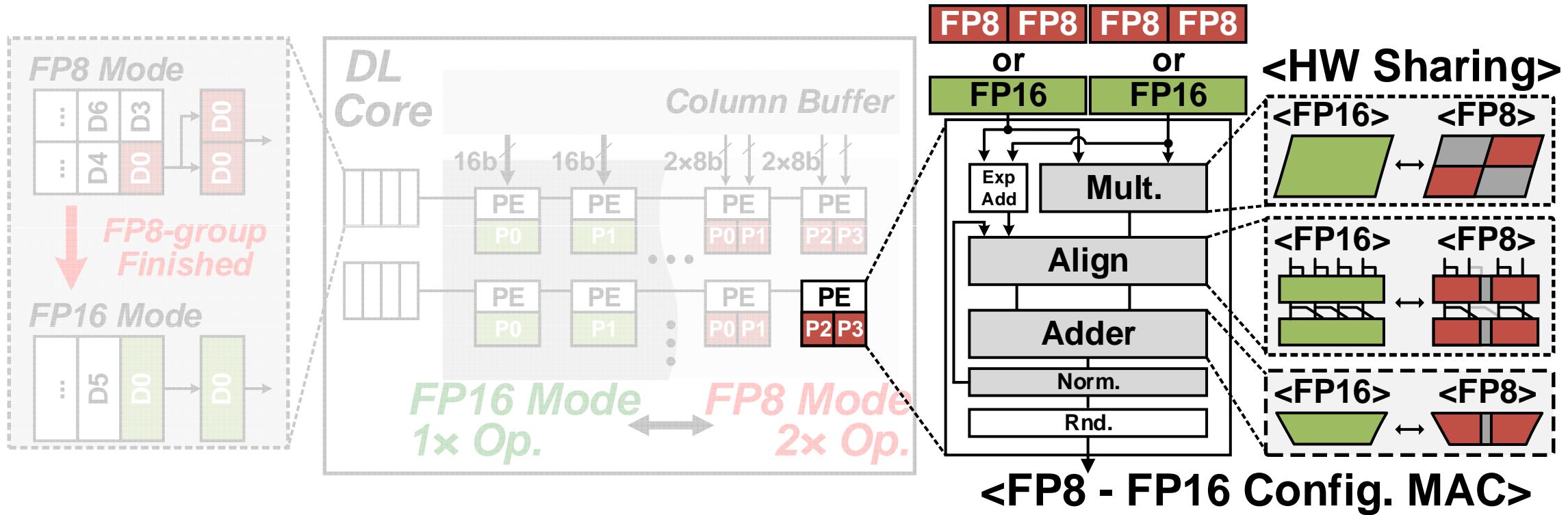
- FP8/FP16 configurable MAC
- **2× throughput for FP8-group, 1× throughput for FP16-group**



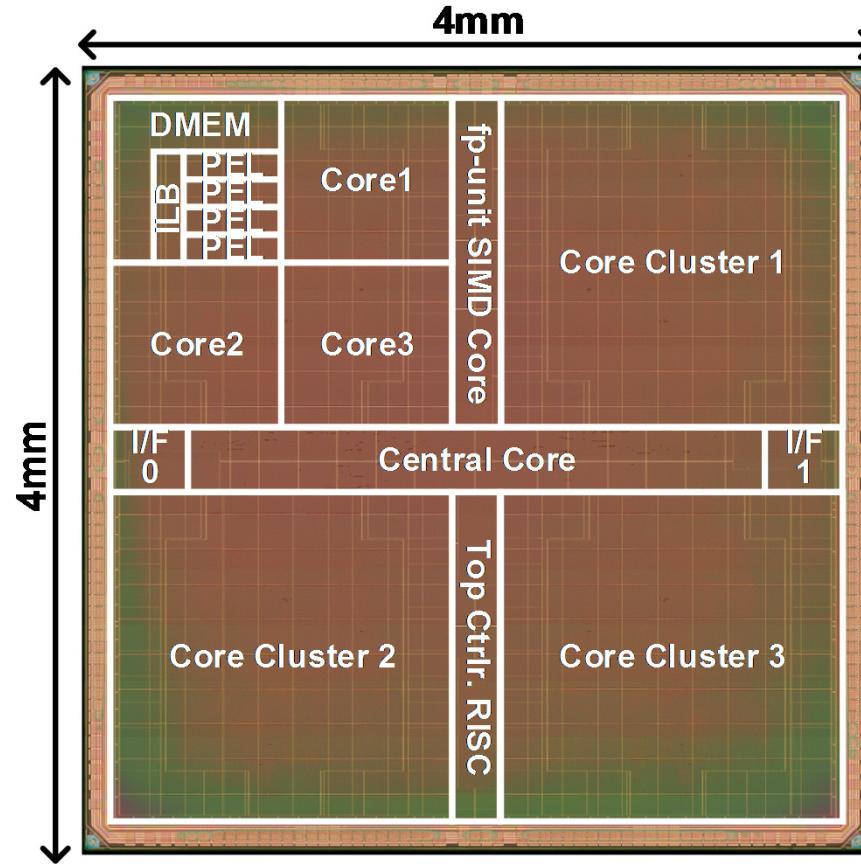
FP8/FP16 Configurable PE

❑ Accelerate FGMP Encoded Input

- FP8/FP16 configurable MAC
- **2× throughput for FP8-group, 1× throughput for FP16-group**



Chip Photograph and Summary

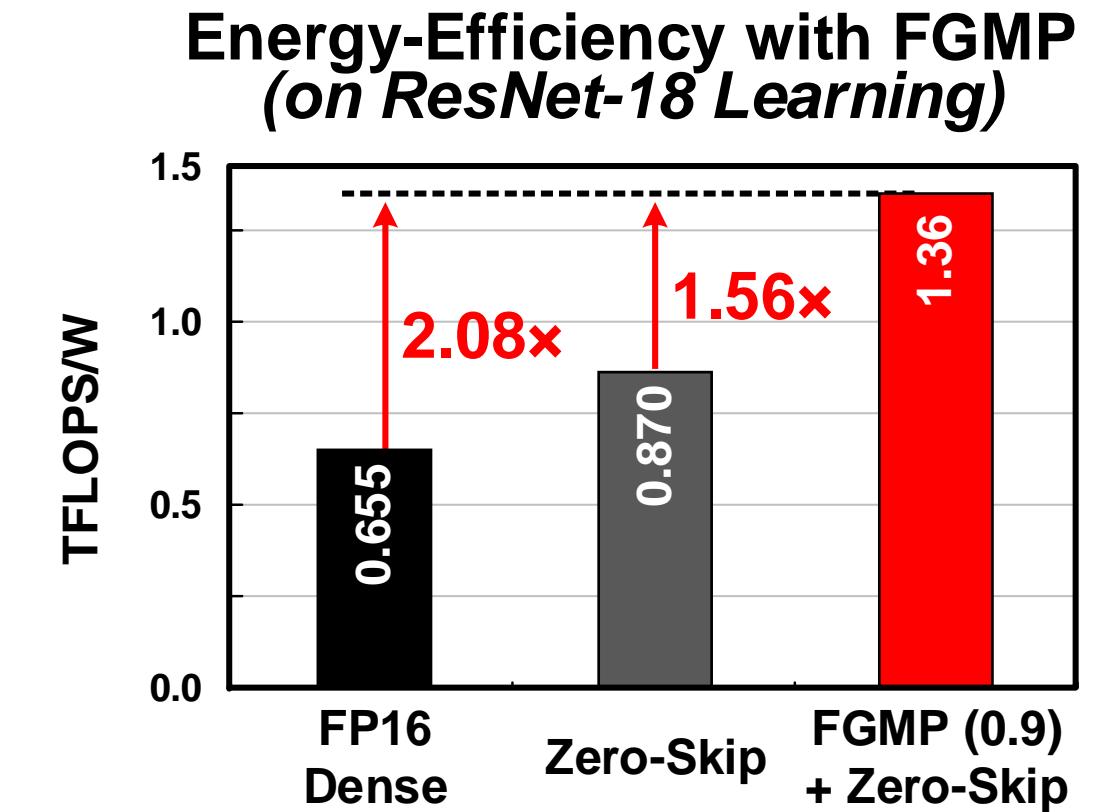
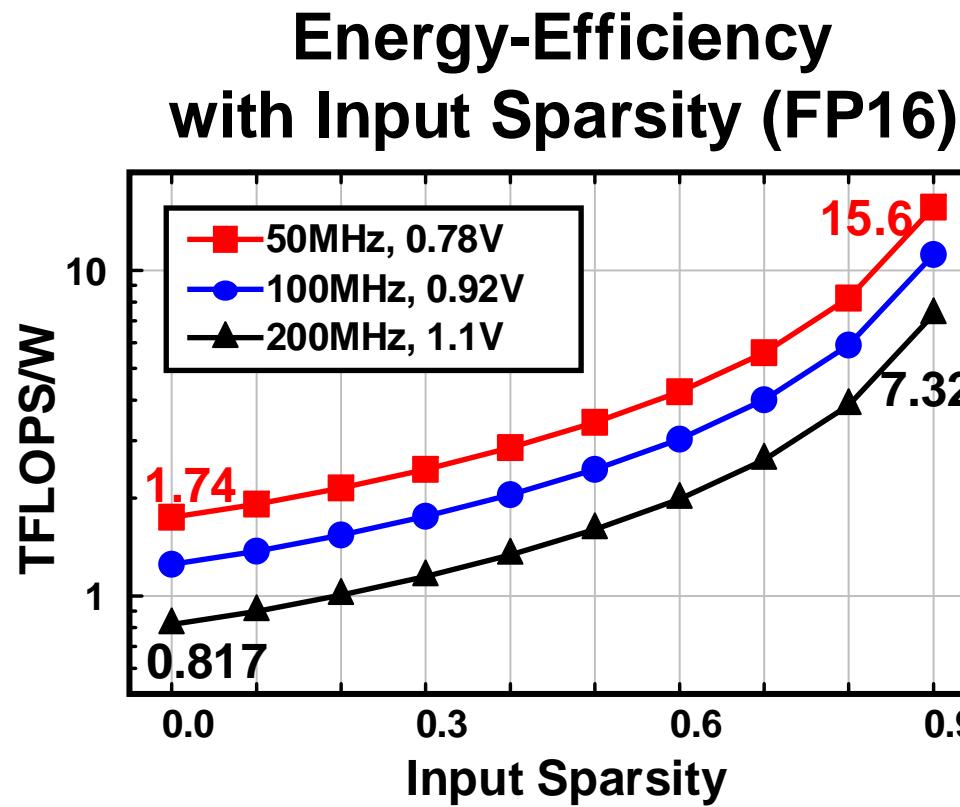


	Specifications	
Technology	65nm 1P8M CMOS	
Die Area	4mm × 4mm (16mm ²)	
SRAM	372 KB	
Supply Voltage	0.78V ~ 1.1V	
Frequency	~ 200MHz	
Data Type	FP8, FP16	
Power Consumption (mW)	43.1mW @ 0.78V, 50MHz	
	367mW @ 1.1V, 200MHz	
Power Efficiency [TFLOPS/W]	FP16	FP8
	50MHz, @0.78V	1.74 -15.6* TFLOPS/W
	200MHz, @1.1V	0.817-7.32* TFLOPS/W
3.48-25.3* TFLOPS/W		
1.63-11.9* TFLOPS/W		

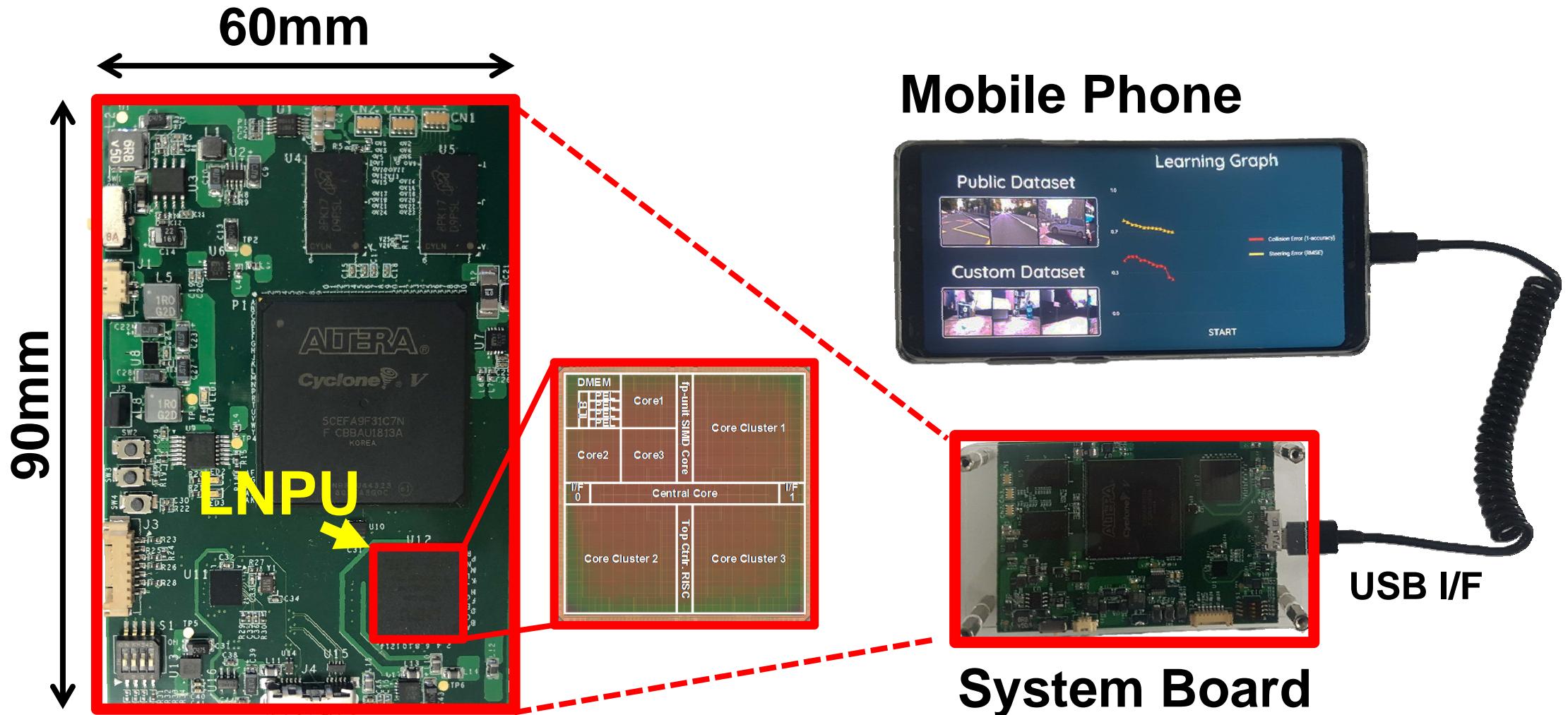
*Effective TFLOPS/W with 90% Input Sparsity

Measurement Results

□ Energy-Efficiency Results



LNPU Verification System

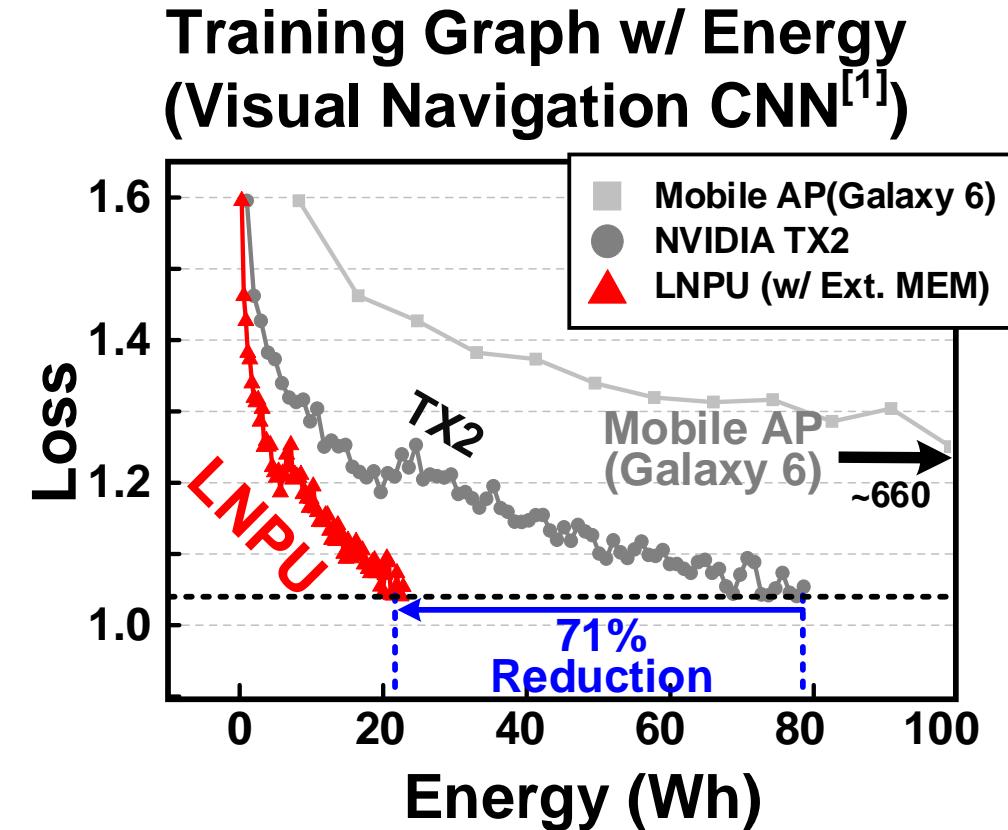
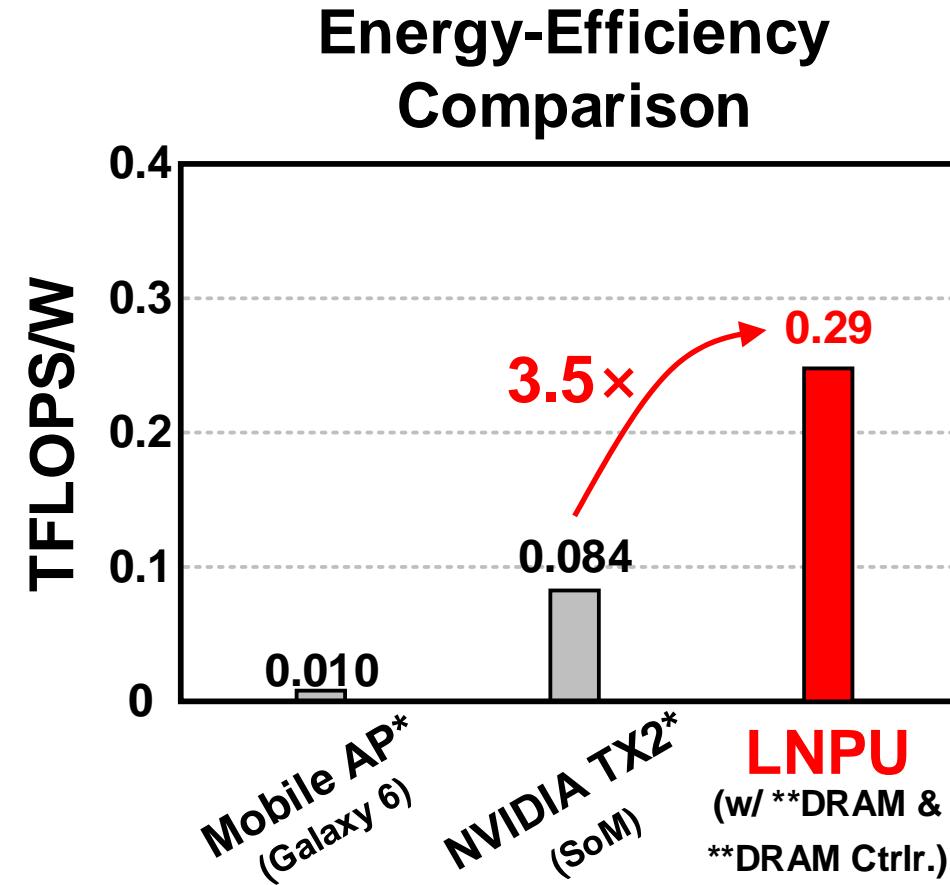


Demonstration Video – DS1

LNPU

**Edge/Mobile CNN Learning
System Demonstration**

Performance Comparison



[1] DroNet, RA-L2018

*for CNN Operation

**Micron, DDR3, 2Gb + Intel PowerPlay, memory controller

7.7: LNPU: A 25.3 TFLOPS/W Sparse Deep-Neural-Network Learning Processor with Fine-Grained Mixed Precision of FP8-FP16

Performance Comparison

	NVIDIA TX2 (Module)	NVIDIA V100 (Module)	S.VLSI 2018 B. Fleischer [6]	<u>This Work</u>
Purpose	GPU	GPU	CNN, FC, RNN	CNN, FC, RNN
DNN Training Support	O	O	O	O
Sparsity Support	Software	Software	X	O (All Learning Step)
Technology [nm]	16	12	14	65
Die Area [mm ²]	-	815	9	16
PE bit-precision [bit]	FP16, FP32	FP16, FP32, FP64	binary, ternary FP16	FP8, FP16, FGMP
Peak Performance GFLOPS	874-1500 (FP16)	120,000(FP16)	1500(FP16)	>300 (FP16) >600 (FP8)
GFLOPS/mm ²	-	5.0** (FP16)	7.7** (FP16)	>18.8
Energy Efficiency [TFLOPS/W]	0.12 (FP16)	0.40 (FP16)	-	1.74(FP16)~3.48(FP8) 15.6*(FP16)~25.3*(FP8)
Power Consumption [mW]	7,500-15,000 (TDP)	300,000 (TDP)	-	43.1 ~ 367

* w/ 90% Input Sparsity ** Normalized for 65nm Tech.

Conclusion

❑ LNPU: An Energy-Efficient DNN Learning Processor

❑ Key Features:

- Fine-grained Mixed Precision

- Reducing 38.9% of EMA (@ResNet-18 Training)

- Fully Reconfigurable Sparse DL Core

- Achieving 1.82× Speed-up (@ResNet-18 Training)

- FP8/FP16 Mixed Operation

- Improving 2.08× Energy-efficiency (@ResNet-18 Training)

A 25.3 TFLOPS/W DNN Learning Processor
for Mobile/Edge Learning